AD-A118 811   MARINE CORPS TACTICAL SYSTEMS SUPPORT ACTIVITY CAMP --ETC F/G 9/2
DIGITAL COMMUNICATIONS TERMINAL HIGH ORDER PROGRAMMING LANGUAGE--ETC(U)
NOV 80 K C SHUMATE, R E SAUER, G E ANDERSON
UNCLASSIFIED 24E005/U-TN-01-VOL-2

NL

1 o 2

AD A118811

3

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER TECHNICAL NOTE 24E005/U-TN-01	2. GOVT ACCESSION NO. AD-A118811	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DIGITAL COMMUNICATIONS TERMINAL HIGH ORDER PROGRAMMING LANGUAGE STUDY (DCT HOL STUDY). VOLUME TWO: APPENDICES		5. TYPE OF REPORT & PERIOD COVERED Final report, Vol. 2
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Maj K.C. SHUMATE USMC Mr. R.E. SAUER Maj G.E. ANDERSON USMC Maj J.W. HOOPER USMC Capt B.F. BRADY USMC		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Marine Corps Tactical Systems Support Activity Marine Corps Base Camp Pendleton, CA 92055		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS DCT Project
11. CONTROLLING OFFICE NAME AND ADDRESS Marine Corps Development and Education Command Quantico, VA 22134		12. REPORT DATE 26 November 1980
		13. NUMBER OF PAGES 100
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Marine Corps Tactical Systems Support Activity Marine Corps Base Camp Pendleton, CA 92055		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for Public Release; distribution unlimited.

DTIC
ELECTE
SEP 0 2 1982
S D
E

18. SUPPLEMENTARY NOTES

See also Volume 1.

Copy available to DTIC does not permit fully legible reproduction

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

HIGH LEVEL LANGUAGE	COMPUTER PROGRAMS
DIGITAL COMMUNICATIONS TERMINAL (DCT)	DCT HOL STUDY
HIGH ORDER PROGRAMMING LANGUAGE	MICROPROCESSORS
NSC800 MICROPROCESSOR	DECISION MAKING
C PROGRAMMING LANGUAGE	DELPHI

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper reports the results of a study to select a high order programming language for the development of computer programs for the digital communications terminal. All languages suitable for use with the NSC800 microprocessor were considered. The nine final candidates were evaluated by a methodology including benchmarking and determination of a figure of merit. During the conduct of the study it became clear that the program support environment must include both a minicomputer software engineering host, and a microcomputer development

DTIC FILE COPY

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT

system. The language selected is Interactive Systems µC. The system includes a cross-compiler running on a PDP-11 and generating code for the NSC800.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

TECHNICAL NOTE 24E005/U-TN-01

DIGITAL COMMUNICATIONS TERMINAL HIGH ORDER PROGRAMMING LANGUAGE STUDY

(DCT HOL STUDY)

VOLUME TWO: APPENDICES

26 NOVEMBER 1980

MARINE

CORPS

TACTICAL

SYSTEMS

SUPPORT

ACTIVITY

CAMP PENDLETON CALIFORNIA 92055

Approved for public release; distribution unlimited.

82 08 31 022

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Accession For

NTIS GRA&I ☒
DTIC TAB □
Unannounced □
Justification

By
Distribution/
Availability Codes

Dist	Avail and/or Special
A	23

DTIC
COPY
INSPECTED
2

APPENDIX A
BIBLIOGRAPHIC INFORMATION

This appendix contains four tabs. Tab 1 presents the documents and articles which were of particular value in the preparation of the report. Tab 2 lists periodicals which were reviewed for advertising and general market information regarding available HOLs. Tab 3 is a detailed bibliography of the trade journal literature on high order languages Tab 4 is a list of the publications obtained from language vendors which are references for the languages which were evaluated.

Tab 1

Primary References

(a) Cherlin, Mokurai, "High-Level Languages for Microcomputers," Mini-Micro Systems, April 1980, pp. 92-103.

(b) Considerations for a Microprocessor Policy, with appendixes, MCTSSA, 18 June 1980.

(c) Harvany, Jozsef and Jozsef Janos, "Software Products for Manufacturing, Design, and Control," Proceedings of the IEEE, Vol. 68, No. 9, September 1980, pp. 1050-1053.

(d) Meyers, Edith, "Software Slump in Sight?" Datamation, May 1980, pp. 93-94.

(e) "More and Better Operating Systems Turn More Chips into Better µCs," Electronics, Vol. 28, No. 6, 15 March 1980, pp. 223-224.

(f) Nichols, J. E., The Structure and Design of Programming Languages, Addison-Wesley, 1975, p. 8.

(g) Pratt, T. W., Programming Languages: Design and Implementation, Prentice-Hall, 1975, p. 207.

(h) Reghizzi, Stefano Crespi, "A Survey of Microprocessor Languages," Computer, Jan 1980, p. 48.

(i) Sabin, M.A., "Portability -- Some Experiences with FORTRAN," Software Practice Experience, Vol. 6, July - September 1976, pp. 393-396.

(j) Schindler, Max, "Pick a Computer Language That Fits the Job," Electronics, 19 July 1980, pp. 62-78.

(k) Schindler, Max, "UNIX with Workbench Will Serve Superminis," Electronics, 8 November 1979, pp. 30-31.

(l) Stiefel, Malcom L., "A Guide to Tool Selection," Mini-Micro Systems, August 1980, pp. 68-76.

Tab 2

Periodicals

1. Computer Design: The Magazine of Computer-Based Systems, Vol 19, No. 5, May 1980.

2. Computer Design: The Magazine of Computer-Based Systems, Vol 19, No. 6, June 1980.

3. Computer Design: The Magazine of Computer-Based Systems, Vol 19, No. 7, July 1980.

4. Electronic Design, Vol. 28, No. 6, March 15, 1980.

5. Electronic Design, Vol. 28, No. 12, June 7, 1980.

6. Electronic Design, Vol. 28, No. 13, June 21, 1980.

7. Electronic Design, Vol. 28, No. 14, July 5, 1980.

8. Electronic Design, Vol. 28, No. 16, August 2 1980.

9. Electronics: The International Magazine of Electronic Technology, Vol. 53, No. 9, April 17, 1980.

10. Electronics: The International Magazine of Electronic Technology, Vol. 53, No. 12, May 22, 1980.

11. Electronics: The International Magazine of Electronic Technology, Vol. 53, No. 13, June 5, 1980.

12. Electronics: The International Magazine of Electronic Technology, Vol. 53, No. 16, July 17, 1980.

13. Electronics: The International Magazine of Electronic Technology, Vol. 53, No. 17, July 31, 1980.

14. Mini-Micro Systems, Vol. XIII, No. 4, April 1980.

15. Mini-Micro Systems, Vol. XIII, No. 6, June 1980.

16. Mini-Micro Systems, Vol. XIII, No. 7, July 1980.

Tab 3

General Bibliography

The information to prepare this bibliography of high order languages was obtained by researching the journals listed below, representing the period March through September 1980:

<u>Byte</u>
<u>Communications of the ACM</u>
<u>Computer Design</u>
<u>Datamation</u>
<u>EDN</u>
<u>Electronics</u>
<u>Electronics Design</u>
<u>Mini-Micro Systems</u>

High Order Languages -- General Information: Journal articles that contain surveys, comparisons, discussion and general information about the various high order Languages.

High Order Languages -- Specific: Journal articles that contain information about specific language characteristics and their use in applications programming.

Microcomputer Development Systems: Journal articles that contain information about the hardware and software characteristics of the various state-of-the-art MDSs that are currently available.

In Circuit Emulation: Journal articles that discuss the uses and types of in-circuit emulator as a product-development and interactive debugging facility.

High Order Language Philosophy: A collection of material by several authors discussing the relative merits of using HOLs for applications development.

High Order Language Study Bibliography

High Order Languages; General Information

1. Cherlin, Modurai, "High-level Languages for Microcomputers," Mini-Micro Systems, Apr 1980, p.89 .

2. Feldman, Jerome A., "High Level Programming for Distributed Computing," Communications of the ACM, Vol 22, Jun 1979, p.353.

3. Hughes, Philip, "Macro Assembers for Micros," Mini-Micro Systems, Apr 1979, p.71.

4. Kroeker, Edwin J., "Multilingual Software Cuts Development Costs," Mini-Micro Systems, May 1980, p.163.

5. Morris, Robert A., "Comparison of Some High-Level Languages," Byte, Feb 1980, p.128.

6. Ogdin, Carol Ann, "The Many Choices In Development Languages," Mini-Micro Systems, Aug 1980, p.81.

7. Reghizzi, Stefano Crespi, "A Survey of Microprocessor Languages," Computer, Jan 1980, p.48.

8. Schindler, Max, "Check Operating-System Features To See How Powerful a uC Will Be," Electronic Design, Nov 22, 1979, p.133.

9. Schindler, Max, "Pick A Computer Language That Fits The Job," Electronic Design, Jul 19, 1980, p.62.

10. Wecker, D.B., R.L., Krutz, & D.T., Tuma, "High Level Design Language Develop Low Level Microprocessor - Independent Software," Computer Design, Jun 1979, p.140.

Product Advertisement

11. Scientific Enterprises, Inc., "The Software Synthesizer; Lets the Engineer Take a Hardware Approach to Program Development," Mini-Micro Systems, Jun 1980, p.45.

High Order Language Study Bibliography

High Order Languages; Specific

ADA

1. Glass, Bob, "From PASCAL to Pebbleman and Beyond," Datamation, Jul 1979, p.146.

APL

2. Gull, W.E. & M.A., Jenkins, "Recursive Data STructures in APL," Communications of the ACM, Vol 22, Jan 1979, p.79.

FORTH

3. Hicks, Stephen M., "FORTH's Forte is Tighter Programming," Electronics, Mar 15, 1979, p.114.

4. Mannoni, Michel, "Forth - An Extensible Path to Efficient Programs," Electronic Design, Jul 19, 1980, p.175.

5. James, John S., "What is FORTH? A Tutorial Introduction," Byte, Aug 1980, p.100.

6. Harris, Kim, "FORTH Extensibility, or how to Write a Compiler in 25 Words or Less," Byte, Aug 1980, p.164.

7. Miller, Richard & Jill Miller, "Breakforth into FORTH!", Byte, Aug 1980, p.150.

8. Moore, Charles H., "The Evolution of FORTH, an Unusual Language," Byte, Aug 1980, p.76.

9. Williams, Gregg, "FORTH Glossary," Byte, Aug 1980, p.186.

MDL - Modular Development Language

10. Schnabel, Dennis, "MDL/u - A New Language for Effective uP Software Development," Electronic Design, Sep 13, 1979, p.102.

MPL - Developed by Motorola

11. Waller, Larry, "Microcomputers to Run Test Hours," Electronics, Jul 19, 1979, p.92.

PASCAL

12. Conrad, Marvin, "PASCAL - A High Level Language for Micros & Minis," Datamation, Jul 1979, p.153.

13. Doty, Keith L., "A Top Down Evaluation of Pascal," Computer Design, May 1980, p.167.

14. Fletcher, Dennis, "PASCAL Power," Datamation, Jul 1979, p.142.

15. Hemenway, Jack & Edward Teja, "PASCAL Update," EDN, Apr 5, 1980, p.100.

16. Mundie, David A., "PASCAL and the Great Race," Byte, Sep 1980, p.94.

17. Shillington, Keith, "Structure: The Key to Pascal's Problem-Solving Power," Datamation, Jul 1979, p.151.

PL/1

18. Marshall, Martin, "PL/1 Seeks Popularity," Electronics, Apr 24, 1980, p.102.

PLZ

19. Riggins, Chris, "When No Single Language Can Do the Job, Make it a Language-Family Matter," Electronic Design, Feb 15, 1979, p.86.

PILOT/P

20. Mundie, David, "PILOT/P: Implementing a High-Level Language in a Hurry," Byte, Jul 1980, p.154.

ROSETTA

21. Warren, Scott K. & Abbe, Dennis, "Presenting ROSETTA Smalltalk," Datamation, May 1980, p.145.

TINY "C"

22. Kern, Christopher O, "A User's Look at Tiny - C," Byte, Dec 1979, p.196.

Product Advertisements

23. Analog Devices Inc., "A PL uS (A Programming Language for Microprocessor Systems)," Electronics, Nov 8, 1979, p.41.

24. Vanguard Systems Corp, "APL/DTC Desktop Microcomputer using APL," Computer Design, Aug 1979, p.190.

25. Whitesmiths', "Whitesmiths' full C Compiler," Electronics, Sep 13, 1979, p.199.

26. Micro Focus, Ltd., "COBOL for 8 & 16 Bit Microcomputers," Mini-Micro Systems, Mar 1980, p.79.

27. Systems Engineering Laboratories Inc., "FORTRAN Compiler (Surpasses the ANSI 77 Standard)," Electronics, Aug 16, 1979, p.155.

28. Bally Corp., "GRAFIX, A Self Teaching, User-Expandable Language," Electronics, Jan 18, 1979, p.48.

29. SofTech Microsystems, "PASCAL Confusion Explained," Datamation, May 1980, p.98.

30. Control Systems Inc., "PASCAL resides in PROM; UCSD Pascal Firmware for 6800 and 6809 Processors Eliminates Interpreter Software," Electronics, Apr 26, 1979, p.212.

31. John Wiley & Sons, Inc. (Book Publishers), "An Introduction to Problem Solving & Programming with PASCAL & PL/1 Structured Programming, 2nd Ed.," Communications of the ACM, Jun 79.

32. Zilog, Inc., "Structured Z80 PASCAL for Microcomputers," Computer Design, Sep 1979, p.166.

33. Digital Research, "PL/1 Shrinks to fit Microprocessors," Electronics, Apr 24, 1980, p.41.

34. Mostek, "PL/M Development Languages," Electronics, Jan 4, 1979, p.33.

35. General Health, "SIMPL, A High Level Interactive Microprocessor Language," Electronics, Jan 17, 1980, p.40.

High Order Language Study Bibliography

Microcomputer Development Systems

1. Bailey, Chris & Tracy Kahl, "Evaluation Delay Cuty By Low-Cost Microprocessor Development Tool," Electronics, Aug 30, 1979, p.121.

2. Gladstone, Bruce E, "Comparing Microcomputer Development System Capabilities," Computer Design, Feb 1979, p.83.

3. Lee, Edwin, "Debunking The Development System Myth (Opinion)," Mini-Micro Systems, Aug 1980, p.107.

4. McCracken, David, "Hybrid Tool for Universal Microprocessor Development," Computer Design, Apr 1980, P.199.

5. McLeod, Jonah, "uC Development Systems Have The Hardware, Software 16-Bit uP's Need," Electronic Design, Sep 1979, P.92.

6. Stiefel, Malcolm L., "A Guide To Tool Selection," Mini-Micro Systems, Aug 1980, p.68.

7. Stiefel, Malcolm L., "Microcomputer Development Systems (Survey)," Mini-Micro Systems, Sep 1979, p.74.

8. Stock, Bruce, "A Centralized Design Support Center," Mini-Micro Systems, Aug 1980, p.87.

Product Advertisements

9. Microsoft, "MS-Pascal Compiler, Optimizer & Modular Code Generators for 8080, Z80, 8086, A8000, and a Pseudo Machine," Computer Design, Aug 1980, p.152.

10. Phoenix Digital Corp, "EXOR 80/XASM Assembles Z80 and 8080 Source Programs into Machine Code," Computer Design, Dec 1979, p.132.

11. Infosoft Systems, Inc. "Microcomputer OS for 8080, 8085, and Z80 based Microcomputers, the I/OS Disk Operating System," Datamation, Unk, p.260.

12. Pascal Development Company, "PASCAL & 8002 Combine in Microprocessor Development," Electronics, Jul 5, 1979, p.206.

13. Danfysik A/S, "The 'UNIQUE' Microprocessor Development System," Electronics, Jul 5, 1979, p.76.

14. Boston Systems Office, "Universal Microprocessor Development System (UMDS)," Electronics, Aug 30 1979, p.226.

High Order Language Study Bibliography

In Circuit Emulation

1. Kelly, James M., "Cut Hardware, Software Development Costs-Take Advantage of In-Circuit-Emulators," Electronic Design, Aug 16, 1979, p.66.
2. Moon, Jim, "Microcomputer For Emulation Bares Hidden Buses, Functions," Electronics, Jul 17, 1980, p.126.
3. Saponas, Tom, "Development System: Soft Keys + Four Buses = Easy Use and Full-Speed Emulation," Electronic Design, Sep 27, 1980, p.38.

Product Advertisements

4. Applied Microsystems, "EX-80 a Z80 Emulator Frees MDS By Taking On Test Chores," Electronics, Apr 12, 1979, p.240.
5. Tecma Inc., "The Microsystem Emulator; A Versatile In-Circuit Emulator Complements Any uP Development System," EDN, Sep 20, 1979, p.71.

High Order Language Study Bibliography

High Order Language Philosphy

1. Bai, Subhash, Yoav Lavi, Asher Kaminker, and Avram Menachem, "Optimising Microprocessor Performance," Mini-Micro Systems, Jun 1980. p.103.

2. Brooks, Ruven E., "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology," Communications of the ACM, Vol 23, Apr 1980, p.207.

3. Caffin, R.N., "The Open Channel," Computer, Mar 1979, p.108.

4. Caudill, Pat, "Using Assembly Coding to Optimize HIgh-Level Language Programs," Electronics, Feb, 1979, p.121.

5. Ehram, John R. "The New Tower of Babel," Datamation, Mar 1980, p.157.

6. Esterling, Bob, "Software Manpower Costs: A Model," Datamation, Mar 1980, p.164.

7. Heckel, Paul, "Developing Software or Microprocessor-Based Products," Mini-Micro Systems, Feb 1980, p.111.

8. Hug, Richard & Dr. Presser, Leon, "Designing Transportable Software," Mini-Micro Systems, May 1980, p.122.

9. Posa, John G., "Microprocessors and Microcomputers," Electronics, Oct 25, 1979, p.144.

10. Posa, John G., "Programming Microcomputer Systems with High-Level Languages," Electronics, Jan 18, 1979.

11. Sheppard, Sylvia B., Curtis, Bill, Milliman, Phil, & Love, Tom, "Modern Coding Practices and Programmer Performance," Computer, Dec 1979, p.41.

12. Turner, Joshua, "The Structure of Modular Programs," Communications of the ACM, Vol 23, May 1980, p.272.

13. Wulf, William A., "Trends in the Design and Implementation of Programming Languages," Computer, Jan 1980, p.14.

14. Klatt, Garth, "PL/1 Vs. PASCAL (Letter to the ED)," Datamation, Jan 1980, p.35.

15. Amort, Anthony, "APL Enthusiast (Letter to the ED)," Datamation, Mar 1980, p.39.

High Order Language Study Bibliography

Miscellaneous Material

1. Allison, Andrew, "Setting Standards for Microprocessors," Mini-Micro Systems, Oct 1979, p.66.

2. Dwyer, Ed, Michel Lafortune, & Michel Bertrand, "Bootstrap Lets Multibus Tap CP/M-Based Software," Electronic Design, May 10, 1980, p.219.

3. Grappel, Rober & Jack Hemenway, "The MC68000 - A 32-bit uP Masquerading as a 16-bit Device," EDN, Feb 20, 1980, p.127.

4. Lowe, Linda, "System for Terminals Creates 'Keyboards' Anyone Can Use," Electronics, Jun 5, 1980, p.39.

5. Moore, Cecil A., "Ready-made Multiplexer Simplifies Multitasking," Electronic Design, Jul 19, 1980, p.153.

Product Advertisements etc.

6. Electro, "Summation of Electro/80 Conference & Exhibition," Electronics, Apr 24, 1980, p.142.

7. Western Digital Corporation, "Chip Set and Development Computer Execute PASCAL Object Programs," Computer Design, May 1979, p.250.

8. Texas Instruments Inc., "TI To Add PASCAL to its FS990 Development Unit," Electronics, Mar 26, 1979, p.36.

9. Advanced Micro Computers, "Z8000 Gets Development Unit, Especially Designed for 16-bit Microprocessor, Unit supports 8-bit Processors Too," Electrcnics, Mar 29, 1979, p.138.

10. Mackintosh Publications Ltd., "Microcomputer Analysis Report indicating the Major Growth in Microprocessor Sales in the next few years will probably be in the area of 16-bit Microprocessors," Electronics, Sep 17, 1979, p.261.

11. National Semiconductor Corp., "Announces the NSC800 Chip; Which Is Built Around the Architecture of Intel Corp.'s 8085 and executes the Instruction Set of Zilog Inc.'s Z80," Electronics, Oct 11, 1979, p.46.

12. Monolithic Systems Corp., "MSC 8004; A Multibus/Z80A Based Microcomputer that Offers 32-Bit Floating Point Arithmethic," Computer Design, Mar 1979, p.210.

Tab 4

Language Documentation

The following documents have been ordered to support this study effort and have been received. They are listed in alphabetical order according to title.

ACT Assembler User's Manual, Version 1.0, Sorcim, June 1980.

Application Note Index, Hewlett-Packard, nd.

"BC -- An Arbitrary Precision Desk-Calculator Language," Lorinda Cherry and Robert Morris, Bell Laboratories, nd.

BDSC User's Guide Addenda v1.32 Edition, Leor Zolman, BD Software, 1980.

BD Software C Compiler v1.3 User's Guide, Leor Zolman, Lifeboat Associates, 1979.

Beginner's Guide for the UCSD PASCAL System, Kenneth L. Bowles, Byte Books, 1980.

C Compiler Systems Interface Manual for 8080 Users, Whitesmiths, Ltd., 1980.

C Compiler User's Manual, Whitesmiths, Ltd., 1980.

C Compiler v1.3 User's Guide, Zolman, Leor, BD Software, 1979.

"Computer Modules," Product Brief, Zilog, Inc., May 1980.

Context Editor for the CP/M Disk System User's Manual, A, Digital Research, 1980.

The C Programming Language, Brian W. Kerninghan and Dennis M. Ritchie, Prentice-Hall, 1978.

CP/M Assembler User's Guide, Digital Research, 1978.

CP/M Dynamic Debugging Tool (DDT) User's Guide, Digital Research, 1978.

CP/M Features and Facilities, An Introduction to, Digital Research, 1980.

CP/M MAC Macro Assembler Language Manual and Applications Guide, Digital Research, 1980.

CP/M SID Symbolic Instruction Debugger User's Guide, Digital Research, 1978.

CP/M 2.0 Interface Guide, Digital Research, 1979.

CP/M 2.0 User's Guide, Digital Research, 1979.

CP/M 2.2 Alteration Guide, Digital Research, 1979.

CP/M ZSID Symbolic Instruction Debugger, Z-80 Version, Digital Research, 1979.

C Reference Manual, Dennis M. Ritchie, Bell Telephone Laboratories, 1975.

CR Z80: BSO Relocating Cross Assembler (with Cross Reference), The Boston Systems Office, Inc., 24 July 1980.

"DC -- An Interactive Desk Calculator," Bell Laboratories, nd.

EDIT 79: A STOIC II Programming Example, Jeffrey L. Zurkow, Avocet Systems, Inc., 1979.

8080/8085 Assembly Language Programming, Intel Corp., 1979.

Electronic Instruments and Systems: 1980, Hewlett-Packard, nd.

"FORTH: A Cost Saving Approach to Software Development," Elizabeth D. Rather and Charles H. Moore, FORTH, Inc., 1976.

FORTRAN-77 User's Manual, SofTech, 1980.

FORTRAN-80 Reference Manual, MicroSoft, 1979.

FORTRAN-80 User's Manual, MicroSoft, 1979.

"FORTRAN-80 8080/8085 ANS FORTRAN-77 Intellec Resident Compiler," Intel Corp., 1978.

FORTRAN IV, Cromemco, Inc., 1979.

FORTRAN Lanuage Manual, Zilog, Inc., September 1979.

FORTRAN User Guide, Revision A, Zilog, Inc., October 1978.

FORTRAN User Reference Manual, SofTech, 1980.

Getting Started with PASCAL/64000, Hewlett-Packard, nd.

GNAT System 10 Operator's Handbook, GNAT Computers, 1980.

GNAT System 10 Reference Manual, Version 1.2, GNAT Computers, March 1980.

A Guide to Intellec Microcomputer Development Systems, Daniel D. McCracken, Intel Corp., 1978.

A Guide to PL/M Programming for Microcomputer Applications, Daniel D. McCracken, Addison-Wesley, 1978.

INFOSOFT Assembler Manual, Infosoft, 1979.

"Interactive display Terminal (IDT) Users' Manual," Litton Data Systems, September 1977.

Intersystems Pascal/Z: A High Level Programming Language, Intersystems, 1980.

Introduction to RIO Text Processing, Zilog, Inc., July 1979.

An Introduction to STOIC, Jonathan M. Sachs, Biomedical Engineering Center for Clinical Instrumentation, March 1978.

Intellec Series II Microcomputer Development System Hardware Reference Manual, Intel Corp., 1980.

Introduction to Microprocessor Components Using PL/Z, Conway et al, Winthrop, 1979.

ISIS-II FORTRAN-80 Compiler Operator's Manual, Intel Corp., 1978.

Link-80 Operator's Guide, Digital Research, April 1980.

"Logic Analyzer Model 1615A," Technical Data, Hewlett-Packard, 15 Mar 1980.

"Logic State Analyzer for Microprocessor Based Systems: Model 1611A," Technical Data, Hewlett-Packard, 15 December 1979.

"Logic State Analyzer for Mini/Micro Computer and Random Logic Analysis: Models 1610A/B," Technical Data, Hewlett-Packard, 15 June 1979.

"A Manual for the Tmg Compiler-Writing Language," M.D. McIlroy, Bell Laboratories, 13 September 1972.

"MCZ-1/50: Z80 Microcomputer Systems," Product Brief, Zilog, Inc., November 1979.

"MCZ-1/70: Z80 Microcomputer System," Product Brief, Zilog, Inc., November 1979.

Microcomputer Components Data Book, Zilog, Inc., February 1980.

Microsoft Utility Software, MicroSoft, 1978.

"The MG Macro Processor," Andrew D. Hall, Bell Telephone Laboratories, Inc., June 1971.

"NROFF Users' Manual," Joseph F. Ossanna, Bell Laboratories, 11 September 1974.

PASCAL/M User's Reference Manual, Sorcim, 1979.

PASCAL/MT User's Manual, MT Microsystems, 1980.

PASCAL User Guide, Zilog, November 1979.

PASCAL User Manual and Report, Kathleen Jensen and Niklaus Wirth, Springer-Verlag, 1974

PASCAL/Z User's Manual, Jeff Moskow, 1980 (also titled: Intersystem's PASCAL/Z, A High Level Programming Language, Version 3.0).

"PLM-80 High Level Programming Language Intellec·Resident Compiler," Intel Corp., September 1976.

PL/1-80 Applications Guide, Digital Research, 1980.

PL/1-80 Applications Manual, Digital Research, 1980.

PLMX User's Guide, Systems Consultatns, Inc., 1980.

PL/M-80 Programming Manual, Intel Corp., 1978.

PLZ Linker User Guide, Revision A, Zilog, Inc., March 1980.

PLZ Version 3 User's Guide, Zilog, Inc., 1979.

"The Portable C Library (on UNIX)," M. E. Lesk, Bell Laboratories, nd.

Programming in C-A Tutorial, Brian W. Kernighan, Bell Laboratories, 1975.

"PROM User's Manual, Revision A, Zilog, July 1978.

RATFOR-A Preprocessor for a Rational FORTRAN, Brian W. Kernighan, Bell Laboratories, 1975.

RATFOR-80: A High-Level Preprocessor for FORTRAN, Version 1.06, The Software Works, 1 February 1980.

RATFOR Reference Manual, Cromemco, 1979.

Report on the Programming Language PLZ/SYS, Tod Snook, et al, Springer-Verlag, 1978.

RIO File Debugger: Reference Manual, Zilog, June 1979.

RIO Symbolic Debugger Reference Manual, Revision B, Zilog, November 1978.

Series/80 Board Level Computer Starplex Development System Data Book, January 1980.

Software Tools, Briar W. Kernighan and P. J. Plauger, Addison-Wesley Publishing Co., 1976.

"A System for Typesetting Mathematics," B. W. Kernighan and L. L. Cherry, Bell Laboratories, nd.

Systems Data Catalog 1980, Intel Corp., August 1979.

"Tiny PASCAL," SuperSoft, nd.

TSA Rlasm Relocating Linking Macro Assembler and TSA Linka Linking Loader, Tsa Software, Inc., 1979.

Tutorial Introduction to the Unix Text Editor, A, B. W. Kernighan, Bell Laboratories, 1975.

"Typesetting Mathematics -- User's Guide," B. W. Kernighan and L. L. Cherry, Bell Laboratories, nd.

UCSD PASCAL User's Manual, SofTech Microsystems, February 1980.

"UNIX Assembler Reference Manual," Dennis M. Ritchie, Bell Laboratories, nd.

Unix Programmer's Manual, K. Thompson and D. M. Ritchie, Bell Laboratories, 1975.

Using FORTH, Second, Revised Edition, FORTH, Inc., March 1980.

Word Star User's Guide, MicroPro International Corporation, 1980.

"YACC--Yet Another Compiler-Compiler," Stephen C. Johnson, Bell Laboratories, nd.

"ZDS-1 Series," Zilog, Inc., 1977.

Z80 -- Assembly Language Programming Manual, Zilog, Inc, April 1980.

"Z-80 Microcomputer Board Series," Zilog, Inc., 1979.

Z80 Microcomputer Software Programming Guide: Z80 Programming Manual, Mostek Corp., December 1977.

Z80 PLZ Debugging Tool (PDT) User Guide, Zilog, Inc., August 1979.

"Z80-RIO," Product Specification, Zilog, Inc., July 1979.

Z80-RIO Operating System User's Manual, Revision A, Zilog, Inc., September 1978.

Z80-RIO Text Editor User's Manual, Revision B, Zilog, Inc., 1978.

Z80-RIO Relocating Assembler and Linker User's Manual, Zilog, Inc., 1978.

Z8000 CPU Technical Manual, Zilog, Inc., 1980.

"Z8000 Cross-Software Package," Product Brief, Zilog, Inc., May 1980.

"Z8000 Development Module," Product Brief, Zilog, Inc., December 1979.

Product brochures have been received from Dynabyte, Tektronix, Whitesmith Ltd., GNAT computers, Zilog, Avocet Corp., Intel Corp., FORTH, Inc., Heurikon Corp., Infosoft, Hewlett-Packard, Cromemco, MicroSoft, Ontel, Emulogic, Hughes, SuperSoft, and Lifeboat Associates.

APPENDIX B
DCT/HOL STUDY BENCHMARK
PROGRAM LISTINGS

This appendix consists of the DCT high order language benchmark program listings for the surviving candidate languages. The listings are divided into two sections. Tabs 1 through 3 contain the pseudo-code specification for the benchmark program. There are three versions of the specification: a code-only version, a heavily commented version, and an error-seeded version.

Tabs 4 through 13 consist of the actual source code listings for the benchmark program written in the target languages. Page numbers are found on each page in the lower right-hand corner because many of the listings continue beyond margin boundaries.

Tab 1

Program DCT Benchmark

(Commented)

```
*****************************************************************************
*****************************************************************************
*****************************************************************************
***                                                                       ***
***                       O C T   B E N C H M A R K                       ***
***                                                                       ***
***   THE PURPOSE OF THIS OCT BENCHMARK PROGRAM IS TO EVALUATE DIFFERENT  ***
***   MICROCOMPUTER HIGH ORDER LANGUAGES (HOL) FOR THE OCT APPLICATION.   ***
***   THE BENCHMARK IS PART OF A LARGER EFFORT TO PICK THE BEST MICRO-    ***
***   COMPUTER HOL FOR THE OCT.  THE PRIMARY PURPOSE OF THE BENCHMARK     ***
***   PROGRAM ITSELF IS TO OBTAIN DATA ON COMPILE TIME, EXECUTION TIME    ***
***   OBJECT PROGRAM SIZE, AND TO ILLUSTRATE ADVANTAGES AND DISADVANTAGES ***
***   OF IMPLEMENTING THE BENCHMARK IN VARIOUS LANGUAGES.                 ***
***                                                                       ***
***   THE BENCHMARK IS PRESENTED BELOW IN ALGORITHMIC FORM WHICH CLOSELY  ***
***   RESEMBLES PASCAL OR ALGOL.  SOME READERS MAY CONSIDER THE BENCHMARK ***
***   TO BE PRESENTED IN A PROGRAMMING DESIGN LANGUAGE.  AT ANY RATE, THE ***
***   BENCHMARK IS PRESENTED HERE WITH EXTENSIVE COMMENTS TO ASSIST THE   ***
***   PROGRAMMER IN IMPLEMENTING THE BENCHMARK IN DIFFERENT LANGUAGES SUCH ***
***   PASCAL, PL/M, FORTRAN, PL/I, AND RATFOR.                            ***
***                                                                       ***
***   THE PROGRAMMER SHOULD READ THE FOLLOWING COMMENTED BENCHMARK CAREFULLY ***
***   BEFORE IMPLEMENTING THE BENCHMARK IN A PARTICULAR LANGUAGE, BECAUSE ***
***   EXTENSIVE GUIDANCE IS GIVEN.  ALSO, A SEPARATE DOCUMENT, ENTITLED   ***
***   "RULES FOR IMPLEMENTING THE OCT BENCHMARK" SHOULD BE STUDIED PRIOR  ***
***   TO CODING THE BENCHMARK IN A PARTICULAR LANGUAGE.                   ***
***                                                                       ***
***   THIS BENCHMARK WAS DEVELOPED AT THE MARINE CORPS TACTICAL SYSTEMS   ***
***   SUPPORT ACTIVITY, CAMP PENDLETON, CALIFORNIA, 92055.                ***
***                                                                       ***
*****************************************************************************
*****************************************************************************
*****************************************************************************


*****************************************************************************
*                                                                           *
*   CONSTANT, TYPE, AND VARIABLE DECLARATIONS FOR DATA ITEMS  REQUIRED TO BE *
*   GLOBAL IN SCOPE.  FOR LANGUAGES WHICH DO NOT SUPPORT 'TYPE', THIS FUNCTION *
*   SHOULD BE ACHIEVED BY USING 'LITERALLY', 'REPLACE', 'MEANS', OR THE     *
*   LOGICAL EQUIVALENT.                                                     *
*                                                                           *
*****************************************************************************


PROGRAM OCTBENCHMARK

   CONST

      ARRAYSIZE = 125;               (* USED TO CONTROL THE SIZE OF ARRAY1 AND   *)
                                     (* ARRAY2 AS WELL AS THE NUMBER OF LOOPS IN *)
                                     (* THE ARRAY ACCESSING LOOP.                *)

      IOLOOPS = 575;                 (* USED TO CONTROL THE NUMBER OF I/O LOOPS, *)

      OPRACTIONLOOPS = 100;          (* USED TO CONTROL THE NUMBER OF LOOPS IN THE *)
                                     (* OPERATOR ACTION CASE STATEMENT LOOP.     *)
```

```
    TIMINGCONTROL = ???;        (* USED TO CONTROL THE NUMBER OF TIMES THAT    *)
                                (* KERNEL , THE ACTUAL BENCHMARK EXECUTION     *)
                                (* PROCEDURE IS CALLED.  THIS WILL BE ADJUSTED*)
                                (* SO THAT RUN TIMES ARE EASY TO MEASURE.      *)

    IOPORT = ???;               (* THIS PARAMETER IS INSTALLATION-DEPENDENT.   *)
                                (* IT WILL BE CHOSEN SO THAT OUTPUT OCCURS     *)
                                (* ON AN I/O PORT THAT IS NOT CONNECTED TO     *)
                                (* ANYTHING.                                   *)

    NUMBERMSGS = 8Ø;            (* CONTROLS THE NUMBER OF TIMES THE MESSAGE    *)
                                (* PROCESSING LOOP IS EXECUTED.                *)

    TESTBYTE = 85;              (* AN EIGHT-BIT QUANTITY OF ALTERNATING BINARY*)
                                (* ONES AND ZEROES TO TEST OUTPUT, SHIFTING,
                                (* AND ROTATING CAPABILITIES.                  *)

    INTEGER1 = 3ØØ;             (* A 16-BIT INTEGER USED TO TEST INTEGER       *)
                                (* ARITHMETIC CAPABILITIES.                    *)

    INTEGER2 = -15Ø;            (* A 16-BIT INTEGER USED TO TEST INTEGER       *)
                                (* ARITHMETIC CAPABILITIES.                    *)

    BITMASK = 1;                (* USED TO MASK THE RIGHTMOST BIT IN A BYTE    *)

    MSGLENGTH = 8Ø;             (* INCOMING MESSAGES ARE EXPECTED TO BE 8Ø     *)
                                (* CHARACTERS IN LENGTH.                       *)

    BUFFERMAX = 15;             (* ALL BUFFERS ARE 16 CHARACTERS IN LENGTH     *)
                                (* RANGING FROM Ø TO 15, INCLUSIVE.            *)

    STARTCODE = 'S';            (* THE ASCII S DENOTES THE START OF A MESSAGE *)


  TYPE

    BUFFERTYPE = ARRAY[Ø..BUFFERMAX] OF CHAR;
                                (* ALL CHARACTER BUFFERS RANGE  FROM          *)
                                (* Ø TO BUFFERMAX, I.E., FROM Ø TO 15.        *)

  VAR

    TIMINGLOOPER,               (* USED AS AN INDEX FOR THE TIMING LOOP       *)
    LOOPCOUNTER,                (* USED AS A LOOP INDEX IN SEVERAL LOOPS.     *)
    WHILECOUNTER,               (* USED TO CONTROL THE WHILE LOOP.            *)
    OPRACTION,                  (* USED TO CONTROL THE CASE STATEMENT,        *)
                                (* RANGING IN VALUE FROM Ø TO 9.              *)
    INBUFFERPTR,                (* INDEXES INTO THE INPUT BUFFER, INBUFFER.   *)
    OUTBUFFERPTR : INTEGER;     (* INDEXES INTO THE OUTPUT BUFFER, OUTBUFFER.*)
                                (* ALL OF THESE 6 INTEGERS SHOULD BE IMPLE-   *)
                                (* MENTED AS 16-BIT INTEGERS.                 *)

    ARRAY1, ARRAY2 : ARRAY[1..ARRAYSIZE] OF INTEGER;
                                (* BOTH ARRAY1 AND ARRAY2 ARE 16-BIT INTEGER *)
                                (* ARRAYS OF SIZE ARRAYSIZE (125) AND ARE     *)
                                (* USED TO TEST 1-DIMENSIONAL ARRAY ACCESSING*)

    INBUFFER : BUFFERTYPE;      (* A CHARACTER BUFFER  USED FOR INPUT         *)

    NEWCHAR : CHAR;             (* A CHARACTER VARIABLE USED FOR TEMPORARY    *)
                                (* STORAGE AND PARAMETER PASSING.             *)

    MIXEDTYPE : RECORD;         (* A COMPLEX RECORD (OR TABLE), CONSISTING    *)
```

```
      CHARBUFFER:BUFFERTYPE; (* OF A 16 ITEM CHARACTER ARRAY FOLLOWED BY  *)
      INTNUMBER : INTEGER;    (* A 16-BIT INTEGER.                         *)
   END RECORD MIXEDTYPE;

FUNCTION GETBYTE : CHAR;

   ****************************************************************
   *                                                              *
   *    PROLOGUE:  FUNCTION GETBYTE RETURNS A CHARACTER VALUE WHEN *
   *               INVOKED.  IT OBTAINS THE CHARACTER FROM THE IN- *
   *               PUT BUFFER (INBUFFER), AND INCREMENTS THE BUFFER *
   *               INDEX (INBUFFERPTR), CHECKING TO SEE IF IT EXCEEDS*
   *               BUFFERMAX IN SIZE.  IF IT DOES, INBUFFERPTR IS  *
   *               RESET TO ZERO.                                  *
   *                                                              *
   *    INPUTS:    NONE.                                           *
   *                                                              *
   *    OUTPUTS:   GETBYTE, A CHARACTER VALUE.                     *
   *                                                              *
   *    CALLED BY: KERNEL.                                         *
   *                                                              *
   *    CALLS:      NO PROCEDURES OR FUNCTIONS.                    *
   *                                                              *
   ****************************************************************

   BEGIN
      GETBYTE := INBUFFER[INBUFFERPTR];
      INBUFFERPTR := INBUFFERPTR + 1;
      IF INBUFFERPTR > BUFFERMAX THEN INBUFFERPTR := 0;
   END GETBYTE;

PROCEDURE PUTBYTE (INCHAR : CHAR, PUTBUFFER : BUFFERTYPE);

   ****************************************************************
   *                                                              *
   *    PROLOGUE:  PROCEDURE PUTBYTE PUTS ONE CHARACTER INTO THE   *
   *               OUTPUT BUFFER (PUTBUFFER).  IT THEN INCREMENTS  *
   *               THE BUFFER INDEX (OUTBUFFERPTR), CHECKING TO SEE *
   *               IF IT EXCEEDS BUFFERMAX IN SIZE.  IF IT DOES,   *
   *               OUTBUFFERPTR IS RESET TO ZERO.                  *
   *                                                              *
   *    INPUTS:    INCHAR, A CHARACTER VALUE, AND PUTBUFFER, A     *
   *               CHARACTER BUFFER OF SIZE BUFFERMAX, WHICH IS    *
   *               PASSED BY REFERENCE AS AN ARRAY TO TEST COMPILER *
   *               EFFICIENCY IN PASSING WHOLE ARRAYS.             *
   *                                                              *
   *    OUTPUTS:   PUTBUFFER (CHANGED).                            *
   *                                                              *
   *    CALLED BY: KERNEL.                                         *
   *                                                              *
   *    CALLS:     NO PROCEDURES OR FUNCTIONS.                     *
   *                                                              *
   ****************************************************************

   BEGIN

      PUTBUFFER[OUTBUFFERPTR] := INCHAR;
      OUTBUFFERPTR := OUTBUFFERPTR + 1;
      IF OUTBUFFERPTR > BUFFERMAX THEN OUTBUFFERPTR := 0;
   END PUTBYTE;
```

```
PROCEDURE KERNEL;

   ****************************************************************
```

```
*                                                                     *
*   PROLOGUE:  PROCEDURE KERNEL  IS THE KERNEL OF THE DCT             *
*              BENCHMARK.  IT IS CALLED REPEATEDLY FROM THE           *
*              MAIN PROGRAM FROM A FOR LOOP CONTROLLED BY TIMING-     *
*              LOOPER.  UPON INVOCATION, EXECUTION OF KERNEL          *
*              PROCEEDS AS FOLLOWS:  A NUMBER OF OUTPUTS ARE          *
*              PERFORMED WITHIN A FOR LOOP, FOLLOWED BY A NUMBER      *
*              OF INTEGER ARRAY MANIPULATIONS, ALSO CONTROLLED BY     *
*              A FOR LOOP.  NEXT, A SERIAL DATA LINK IS SIMULATED     *
*              BY SEARCHING FOR A START CODE AND THEN INPUTTING       *
*              A 80-CHARACTER MESSAGE.  THIS IS DONE REPEATEDLY       *
*              AS CONTROLLED BY A FOR LOOP.  THE NEXT LOOP SIM-       *
*              ULATES OPERATOR INPUTS, WITH MULTI-PATH BRANCHING      *
*              CONTROLLED BY THE CASE CONSTRUCT, WITH THIS ACTION     *
*              REPEATED A NUMBER OF TIMES SINCE  THE CASE ACTION      *
*              IS NESTED WITHIN A FOR LOOP.  UPON COMPLETION OF       *
*              THIS FOR LOOP, KERNEL  RETURNS CONTROL TO THE          *
*              MAIN PROGRAM.                                          *
*                                                                     *
*   INPUTS:    NONE.                                                  *
*                                                                     *
*   OUTPUTS:   NONE.                                                  *
*                                                                     *
*   CALLED BY: MAIN PROGRAM.                                          *
*                                                                     *
*   CALLS :    FUNCTION GETBYTE, PROCEDURE PUTBYTE.                   *
*                                                                     *
***********************************************************************

(* LOCAL VARIABLE DECLARATIONS *)

VAR

   OUTBUFFER : BUFFERTYPE;   (*        DECLARED LOCALLY             *)
                             (* TO PERMIT PASSING OF AN ARRAY TO    *)
                             (* PROCEDURE PUTBYTE.                  *)

BEGIN

   (* PERFORM I/O OPERATIONS FOR IOLOOP NUMBER OF TIMES.  NOTHING *)
   (* SHOULD BE CONNECTED TO IOPORT TO ELIMINATE PERIPHERAL DEVICE*)
   (* TIMING DEPENDENCIES.                                         *)

   FOR LOOPCOUNTER := 1 TO IOLOOPS DO
      OUTPUT(NOT(TESTBYTE), IOPORT);

   (* PERFORM INTEGER ARRAY OPERATIONS AS CONTROLLED BY ARRAYSIZE *)

   FOR LOOPCOUNTER := 0 TO ARRAYSIZE DO
      ARRAY1[LOOPCOUNTER] := ARRAY2[ARRAYSIZE - LOOPCOUNTER];

   (* SIMULATE PROCESSING AN INCOMING MESSAGE BY LOOKING FOR A     *)
   (* START CODE; THEN INPUT MSGLENGTH NUMBER OF CHARACTERS        *)
   (* USING GETBYTE TO INPUT EACH CHARACTER AND PUTBYTE TO OUTPUT *)
   (* EACH CHARACTER.  THIS WHOLE PROCESS IS REPEATED NUMBERMSG   *)
   (* NUMBER OF TIMES AS CONTROLLED BY THE FOR LOOP.              *)

   FOR LOOPCOUNTER := 1 TO NUMBERMSGS DO
      BEGIN
         REPEAT  (* UNTIL WE FIND A START CODE *)
            NEWCHAR := GETBYTE;
         UNTIL NEWCHAR = STARTCODE;

         WHILECOUNTER := 1;     (* FOUND A START CODE, INITIALIZE   *)
                                (* WHILECOUNTER TO INPUT THE MESSAGE*)
```

```
          WHILE WHILECOUNTER < MSGLENGTH DO

          (* THE PURPOSE OF THIS WHILE LOOP IS TO TEST A LANGUAGE'S*)
          (*  WHILE CAPABILITY .  IN THOSE LANGUAGES WHICH DO NOT *)
          (* SUPPORT A WHILE CONSTRUCT, THE WHILE LOGIC MUST BE    *)
          (* IMPLEMENTED BY THE STRUCTURED USE OF A GOTO OR OTHER  *)
          (* CONSTRUCT.  ITERATIVE LOOPING IS NOT PERMITTED.        *)

             BEGIN  (* INPUT MESSAGE CHARACTERS *)
                NEWCHAR := GETBYTE;  (* GET NEXT CHARACTER IN MSG  *)

                (* NOW SHIFT THE CHARACTER RIGHT ONE BIT AND SEE IF*)
                (* THE RIGHTMOST BIT IS A ONE.                      *)

                IF (RIGHTSHIFT(NEWCHAR, 1) AND BITMASK) = 1 THEN
                   PUTBYTE(NEWCHAR, OUTBUFFER);
                ELSE (* DO THE SAME THING - THIS IF/ELSE IS TO      *)
                     (* CHECK COMPILER IF/ELSE CAPABILITY ONLY.     *)
                   PUTBYTE(NEWCHAR, OUTBUFFER);
                WHILECOUNTER := WHILECOUNTER + 1;
             END WHILE;
       END; (* FOR LOOPCOUNTER := 0 TO NUMBERMSGS LOOP *)

OPRACTION := 0;     (* INITIALIZE OPRACTION TO ZERO SO THAT IT CAN  *)
                    (* RANGE IN VALUE FROM 0 TO 9.  IN THOSE        *)
                    (* LANGUAGES WHICH SUPPORT A MODULO FUNCTION,   *)
                    (* OPRACTION SHOULD BE IMPLEMENTED  AS A MODULO *)
                    (* OTHERWISE IT SHOULD BE INCREMENTED BY ONE    *)
                    (* EACH LOOP, CHECKED AGAINST 9, AND RESET TO   *)
                    (* ZERO IF GREATER THAN 9 AS DONE BELOW AT THE  *)
                    (* BOTTOM OF THE CASE STATEMENT.                *)

FOR LOOPCOUNTER := 0 TO OPRACTIONLOOPS DO
    BEGIN   (* CASE OPRACTION WHERE OPRACTION RANGES FROM 0 TO 9    *)
       CASE OPRACTION OF

                       (* SHIFT TESTBYTE TO THE RIGHT BY THREE BITS *)
                       (* THEN STORE IN MIXEDTYPE.                   *)
          0,3,9      : MIXEDTYPE.CHARBUFFER[OPRACTION] :=
                          RIGHTSHIFT(TESTBYTE, 3);

          1,4,7      : (* CIRCULAR ROTATE TESTBYTE TO THE LEFT BY 2 *)
                       (* BITS AND CHECK TO SEE IF THE RIGHTMOST    *)
                       (* BIT IS A ONE.  STORE IN MIXEDTYPE.        *)
                       MIXEDTYPE.CHARBUFFER[OPRACTION] :=
                          LEFTROTATE(TESTBYTE, 2) AND BITMASK;

          2,6        : (* MOVE ALL 16 CHARACTERS IN INBUFFER TO     *)
                       (* THE CHARACTER BUFFER PORTION OF MIXEDTYPE *)
                       MIXEDTYPE.CHARBUFFER := INBUFFER;

          OTHERWISE:   (* WILL BE EXECUTED WHEN OPRACTION = 5 OR 8. *)
                       (* TEST LANGUAGE CAPABILITY FOR 16-BIT       *)
                       (* ARITHMETIC.                               *)
                       MIXEDTYPE.INTNUMBER := ((((INTEGER1/INTEGER2)*
                          INTEGER1)/INTEGER2)*INTEGER2) + INTEGER1;

       END CASE;

       OPRACTION := OPRACTION + 1;    (* INCREMENT OPRACTION AND    *)
                                      (* ALLOW IT TO RANGE FROM 0   *)
       IF OPRACTION > 9 THEN          (* TO 9 IN VALUE.             *)
          OPRACTION := 0;
    END; (* FOR LOOPCOUNTER := 0 TO OPRACTIONLOOPS *)

END; (* PROCEDURE KERNEL  *)
```

```
BEGIN    (*************************************************************)
         (*                                                           *)
         (*    MAIN PROGRAM - - EXECUTION BEGINS HERE                 *)
         (*                                                           *)
         (*************************************************************)

   WRITELINE ('BEGIN BENCHMARK EXECUTION');  (* WHEN THIS MESSAGE APPEARS *)
                                             (* ON THE CRT, BEGIN TIMING   *)
                                             (* THE BENCHMARK EXECUTION.   *)
   FOR LOOPCOUNTER := 0 TO ARRAYSIZE DO
      ARRAY2[LOOPCOUNTER] := LOOPCOUNTER;  (* FILL ARRAY2 WITH ASCENDING  *)
                                           (* INTEGERS, 0 - 15.           *)

   FOR INBUFFERPTR := 0 TO BUFFERMAX DO
      INBUFFER[INBUFFERPTR] := 'A';        (* FILL THE INPUT BUFFER WITH  *)
                                           (* PHONY TEXT , ALL A'S.       *)

   INBUFFER[10] := STARTCODE;              (* PUT ONE STARTCODE IN BUFFER *)
                                           (* SO KERNEL WILL FIND IT.     *)

   INBUFFERPTR, OUTBUFFERPTR := 0;         (* POINT BOTH BUFFER INDICES   *)
                                           (* TO FIRST CHARACTER IN BUFFER*)

   FOR TIMINGLOOPER := 0 TO TIMINGCONTROL DO  (* THIS LOOP CONTROLS HOW    *)
      KERNEL;                                 (* MANY TIMES KERNEL IS      *)
                                              (* CALLED, THUS CONTROLLING  *)
                                              (* BENCHMARK EXECUTION TIME  *)

   WRITELINE (MIXEDTYPE, 'END EXECUTION');  (* WHEN THIS MESSAGE APPEARS ON *)
                                            (* THE CRT, STOP TIMING THE     *)
                                            (* BENCHMARK EXECUTION TIME.    *)

END.
```

Tab 2
Program DCT Benchmark (Code Only)

```
PROGRAM UCTBENCHMARK

   CONST

      ARRAYSIZE = 125;                 IOLOOPS = 575;
      OPRACTIONLOOPS = 100;            TIMINGCONTROL = 77;
      IOPORT = 77;                     NUMBERMSGS = 200;
      TESTBYTE = 85;                   INTEGER1 = 300;
      INTEGER2 = -157;                 BITMASK = 1;
      MSGLENGTH = 80;                  BUFFERMAX = 15;
      STARTCODE = 'S';

   TYPE

      BUFFERTYPE = ARRAY[0..BUFFERMAX] OF CHAR;

   VAR

      TIMINGLOOPER, LOOPCOUNTER, WHILECOUNTER, OPRACTION, INBUFFERPTR,
         OUTBUFFERPTR : INTEGER;
      ARRAY1, ARRAY2 : ARRAY[0..ARRAYSIZE] OF INTEGER;
      INBUFFER : BUFFERTYPE;
      NEWCHAR : CHAR;

      MIXEDTYPE : RECORD;
         CHARBUFFER : BUFFERTYPE;
         INTNUMBER : INTEGER;
      END RECORD MIXEDTYPE;

   FUNCTION GETBYTE : CHAR;
      BEGIN
         GETBYTE := INBUFFER[INBUFFERPTR];
         INBUFFERPTR := INBUFFERPTR + 1;
         IF INBUFFERPTR > BUFFERMAX THEN INBUFFERPTR := 0;
      END GETBYTE;

   PROCEDURE PUTBYTE (INCHAR : CHAR, PUTBUFFER : BUFFERTYPE);
      BEGIN
         PUTBUFFER[OUTBUFFERPTR] := INCHAR;
         OUTBUFFERPTR := OUTBUFFERPTR + 1;
         IF OUTBUFFERPTR > BUFFERMAX THEN OUTBUFFERPTR := 0;
      END PUTBYTE;

   PROCEDURE KERNEL;

      VAR OUTBUFFER : BUFFERTYPE;

      BEGIN

         FOR LOOPCOUNTER := 0 TO IOLOOPS DO
            OUTPUT(NOT(TESTBYTE),IOPORT);

         FOR LOOPCOUNTER := 0 TO ARRAYSIZE DO
            ARRAY1[LOOPCOUNTER] := ARRAY2[ARRAYSIZE - LOOPCOUNTER];

         FOR LOOPCOUNTER := 0 TO NUMBERMSGS DO
            BEGIN
               REPEAT
                  NEWCHAR := GETBYTE;
               UNTIL NEWCHAR = STARTCODE;

               WHILECOUNTER := 0;
```

```
          WHILE WHILECOUNTER < MSGLENGTH DO
            BEGIN
              NEWCHAR := GETBYTE;
              IF (RIGHTSHIFT(NEWCHAR, 1) AND BITMASK) = 1 THEN
                PUTBYTE(NEWCHAR, OUTBUFFER);
              ELSE
                PUTBYTE(NEWCHAR, OUTBUFFER);
              WHILECOUNTER := WHILECOUNTER + 1;
          END WHILE;
        END FOR;

    OPRACTION := 0;

    FOR LOOPCOUNTER := 1 TO OPRACTIONLOOPS DO
      BEGIN
        CASE OPRACTION OF

          0,3,9     : MIXEDTYPE.CHARBUFFER[OPRACTION] :=
                         RIGHTSHIFT(TESTBYTE, 3);

          1,4,7     : MIXEDTYPE.CHARBUFFER[OPRACTION] :=
                         LEFTROTATE(TESTBYTE, 2) AND BITMASK;

          2,6       : MIXEDTYPE.CHARBUFFER := INBUFFER;

          OTHERWISE: MIXEDTYPE.INTNUMBER := ((((INTEGER1/INTEGER2)*
                         INTEGER1)/INTEGER2)*INTEGER2) + INTEGER1;

        END CASE;

        OPRACTION := OPRACTION + 1;
        IF OPRACTION > 9 THEN OPRACTION := 0;
      END FOR;
    END KERNEL;

BEGIN  (* PROGRAM EXECUTION *)

  WRITELINE ('BEGIN BENCHMARK EXECUTION');

  FOR LOOPCOUNTER := 0 TO ARRAYSIZE DO
    ARRAY2[LOOPCOUNTER] := LOOPCOUNTER;

  FOR INBUFFERPTR := 0 TO BUFFERMAX DO
    INBUFFER[INBUFFERPTR] := 'A';

  INBUFFER[10] := STARTCODE;
  INBUFFERPTR, OUTBUFFERPTH := 0;

  FOR TIMINGLOOPER := 0 TO TIMINGCONTROL DO
    KERNEL;

  WRITELINE (MIXEDTYPE, 'END EXECUTION');

END.
```

Tab 3

Error-Seeded DCT BENCHMARK

```
PROGRAM OCTBENCHMARK

*************************************************************************
*************************************************************************
*************************************************************************
***                                                                   ***
***        E R R O R   S E E D E D   O C T   B E N C H M A R K        ***
***                                                                   ***
***   THIS VERSION OF THE OCT BENCHMARK CONTAINS FIVE ERRORS DESIGNED ***
***   TO CAUSE COMPILE-TIME ERRORS DUE TO INCORRECT SYNTAX.  HOWEVER, ***
***   NOT ALL COMPILERS WILL RECOGNIZE ALL OF THE ERRORS AT COMPILE TIME.***
***   THE FIVE ERRORS ARE CLEARLY MARKED IN THE LISTING BELOW WITH    ***
***   A SERIES OF ASTERISKS ALONG WITH AN EXPLANATION OF THE ERROR.   ***
***                                                                   ***
*************************************************************************
*************************************************************************
*************************************************************************


   CONST

      ARRAYSIZE = 125;              IOLOOPS = 575;
      OPRACTIONLOOPS = 100;         TIMINGCONTROL = 77;
      IOPORT = 77;                  NUMBERMSGS = 200;
      TESTBYTE = 65;                INTEGER1 = 300;
      INTEGER2 = -150;              BITMASK = 1;
      MSGLENGTH = 80;               BUFFERMAX = 15;
      STARTCODE = 'S';

   TYPE

      BUFFERTYPE = ARRAY[0..BUFFERMAX] OF CHAR;

   VAR

   (*****    ERROR 1: BY PLACING COMMENT BRACKETS AROUND LOOPCOUNTER IN THE *)
   (*****       LINE BELOW, THE DECLARATION OF LOOPCOUNTER IS ELIMINATED.   *)
   (*****       THE PURPOSE IS TO SEE HOW DIFFERENT COMPILERS TREAT         *)
   (*****       VARIABLE DECLARATIONS OR THE LACK THEREOF.                  *)


      TIMINGLOOPER, (* LOOPCOUNTER,*) WHILECOUNTER, OPRACTION, INBUFFERPTR,
         OUTBUFFERPTR : INTEGER;
      ARRAY1, ARRAY2 : ARRAY[0..ARRAYSIZE] OF INTEGER;
      INBUFFER : BUFFERTYPE;
      NEWCHAR : CHAR;

      MIXEDTYPE : RECORD;
         CHARBUFFER : BUFFERTYPE;
         INTNUMBER : INTEGER;
      END RECORD MIXEDTYPE;

   FUNCTION GETBYTE : CHAR;
      BEGIN
         GETBYTE := INBUFFER[INBUFFERPTR];
         INBUFFERPTR := INBUFFERPTR + 1;
         IF INBUFFERPTR > BUFFERMAX THEN INBUFFERPTR := 0;
      END GETBYTE;
```

```
PROCEDURE PUTBYTE (INCHAR : CHAR; PUTBUFFER : BUFFERTYPE);
   BEGIN
      PUTBUFFER[OUTBUFFERPTR] := INCHAR;
      OUTBUFFERPTR := OUTBUFFERPTR + 1;

      (*****   ERROR 2:  IN THE LINE BELOW, BUFFERMAX HAS BEEN REPLACED *)
      (*****      WITH NEWCHAR SO THAT OUTBUFFERPTR, AN INTEGER, IS      *)
      (*****      NOW COMPARED TO NEWCHAR, A CHARACTER QUANTITY.  THE    *)
      (*****      PURPOSE IS TO EVALUATE TYPE-CHECKING OF COMPARISONS.   *)

      IF OUTBUFFERPTR > NEWCHAR THEN OUTBUFFERPTR := 0;
   END PUTBYTE;

PROCEDURE KERNEL;

   VAR OUTBUFFER : BUFFERTYPE;

   BEGIN

      FOR LOOPCOUNTER := 0 TO IOLOOPS DO
         OUTPUT(NOT(TESTBYTE),IOPORT);

      FOR LOOPCOUNTER := 0 TO ARRAYSIZE DO
         ARRAY1[LOOPCOUNTER] := ARRAY2[ARRAYSIZE  - LOOPCOUNTER];

      FOR LOOPCOUNTER := 0 TO NUMBERMSGS DO
         BEGIN
            REPEAT
               NEWCHAR := GETBYTE;
            UNTIL NEWCHAR = STARTCODE;

            WHILECOUNTER := 0;

            WHILE WHILECOUNTER < MSGLENGTH DO
               BEGIN
                  NEWCHAR := GETBYTE;

                  (*****   ERROR 3:  IN THE LINE BELOW, OUTBUFFER HAS *)
                  (*****      BEEN REPLACED WITH ARRAY1.  THE PURPOSE *)
                  (*****      IS TO EVALUATE TYPE-CHECKING OF PARA-   *)
                  (*****      METERS BEING PASSED, SINCE ARRAY1 IS    *)
                  (*****      AN INTEGER ARRAY AND OUTBUFFER  IS A    *)
                  (*****      CHARACTER ARRAY.                        *)

                  IF (RIGHTSHIFT(NEWCHAR, 1) AND BITMASK) = 1 THEN
                     PUTBYTE(NEWCHAR, ARRAY1);
                  ELSE
                     PUTBYTE(NEWCHAR, OUTBUFFER);
                  WHILECOUNTER := WHILECOUNTER + 1;

            (*****   ERROR 4:  THIS ERROR INVOLVES REMOVING AN 'END' *)
            (*****   BY BRACKETING IT AS A COMMENT.  THE PURPOSE IS  *)
            (*****   TO EVALUATE ERROR RECOVERY WITH MISSING         *)
            (*****   BEGIN/END PAIRS.  IN LANGUAGES WITHOUT BEGIN/   *)
            (*****   END PAIRS, THE REMOVAL OF A BRACE OR GOTO WILL  *)
            (*****   SUFFICE AS THE LOGICAL EQUIVALENT.              *)
         (* END WHILE; *)
         END FOR;

   OPRACTION := 0;

   FOR LOOPCOUNTER := 0 TO OPRACTIONLOOPS DO
      BEGIN
         CASE OPRACTION OF

            0,3,9    : MIXEDTYPE.CHARBUFFER[OPRACTION] :=
                    RIGHTSHIFT(TESTBYTE, 3);

          1,4,7     : MIXEDTYPE.CHARBUFFER(OPRACTION) :=
                       LEFTROTATE(TESTBYTE, 2) AND BITMASK;

          2,8       : MIXEDTYPE.CHARBUFFER := INBUFFER;

          (*****    ERROR 5:  IN THE NEXT LINE, A LEFT PARENTHESIS   *)
          (*****      HAS BEEN REMOVED TO CHECK FOR UNBALANCED       *)
          (*****      PARENTHESES ERROR DETECTION AND REPORTING.     *)

          OTHERWISE: MIXEDTYPE.INTNUMBER := (((INTEGER1/INTEGER2)*
                       INTEGER1)/INTEGER2)*INTEGER2) - INTEGER1;

        END CASE;

        OPRACTION := OPRACTION + 1;
        IF OPRACTION > 9 THEN OPRACTION := 0;
      END FOR;
    END KERNEL;

BEGIN  (* PROGRAM EXECUTION *)

  WRITELINE ('BEGIN BENCHMARK EXECUTION');

  FOR LOOPCOUNTER := 0 TO ARRAYSIZE DO
    ARRAY2[LOOPCOUNTER] := LOOPCOUNTER;

  FOR INBUFFERPTR := 0 TO BUFFERMAX DO
    INBUFFER[INBUFFERPTR] := 'A';

  INBUFFER[10] := STARTCODE;
  INBUFFERPTR, OUTBUFFERPTR := 0;

  FOR TIMINGLOOPER := 0 TO TIMINGCONTROL DO
    KERNEL;

  WRITELINE (MIXEDTYPE, 'END EXECUTION');

END.
```

Tab 4

Interactive "C" Source

```
#

/* PROGRAM DCTBENCHMARK */

/* Final version, with direct code. At Interactive 27OCT80  */


#define ARRAYSIZE 125
#define OPRACTIONLOOPS 100
#define IOPORT 32
#define TESTBYTE 85
#define INT2 -150
#define MSGLENGTH 80
#define STARTCODE 'S'
#define IOLOOPS 575
#define TIMINGCONTROL 12
#define NUMBERMSGS 200
#define INT1 300
#define BITMASK 1
#define BUFFERMAX 15

/* Global data */

int timloop, whilcntr, opaction;
int inbufptr, outbfptr;
int array1[ARRAYSIZE+1],array2[ARRAYSIZE+1];
int loopcntr,index;

char inbuffer[BUFFERMAX+1];
char newchar;

char notbyte; /* Global for reference in asm ....  */

struct record1 {
        char charbufr[BUFFERMAX+1];
        int intnumbr;
};
struct record1 mixtype;

/* End of data declarations */


/*************************************************
*                                               *
*                 Main Program                   *
*                                               *
*************************************************/


main()
{
     printf("Begin benchmark execution\n");

     for(loopcntr = 0; loopcntr <= ARRAYSIZE; loopcntr++)
        array2[loopcntr] = loopcntr;

     for(inbufptr = 0; inbufptr <= BUFFERMAX; inbufptr++)
        inbuffer[inbufptr] = 'A';

     inbuffer[10] = STARTCODE;
     inbufptr = outbfptr = 0;
```

```
    for(timloop = 0; timloop <= TIMINGCONTROL; timloop++)
       kernel();

    for(index = 0; index <= BUFFERMAX; index++)
       printf("%c",mixtype.charbufr[index]);

    printf("%d\n",mixtype.intnumbr);
    printf("End execution\n");

} /* End main program **************************************************/

/*Function definitions  */

/* KERNEL: Exercises the support functions.  */

kernel()
{
    char outbufr[BUFFERMAX + 1];

    notbyte = ~TESTBYTE;

    for(loopcntr = 0; loopcntr <= IOLOOPS; loopcntr++)
       {
               asm  lda.nn     _notbyte; /* a <-- byte to be output */
               asm  ldrn       c,IOPORT; /* c <-- output port */
               asm  out.cr     a;        /* do the output  */
       }

    for(loopcntr = 0; loopcntr <= ARRAYSIZE; loopcntr++)
       array1[loopcntr] = array2[ARRAYSIZE - loopcntr];

    for(loopcntr = 0; loopcntr <= NUMBERMSGS; loopcntr++)
    {
         do {
            newchar = getbyte();
         } while(newchar != STARTCODE);

         whilcntr = 0;
         while(whilcntr++ < MSGLENGTH)
         {
            if (((newchar = getbyte() ) >> 1 ) & BITMASK )
               putbyte(newchar,outbufr);
            else
               putbyte(newchar,outbufr);
         } /* End while */

     } /* End for */

     opaction = 0;
     for(loopcntr = 0; loopcntr <= OPRACTIONLOOPS; loopcntr++)
     {
       switch (opaction)
       {
         case 0:
         case 3:
         case 9: mixtype.charbufr[opaction] =
                                   (TESTBYTE >> 3);
                 break;
         case 1:
         case 4:
         case 7: mixtype.charbufr[opaction] =
                  lftrot(TESTBYTE, 2) & BITMASK;
               break;
         case 2:
         case 6: for(index = 0; index <= BUFFERMAX; index++)
                  mixtype.charbufr[index] = inbuffer[index];
               break;
         default: mixtype.intnumbr = ((((INT1/INT2)*
                  INT1)/INT2)*INT2) + INT1;
                break;
      } /* End switch */

      if (++opaction > 9) opaction = 0;
    } /* End for */
} /* End kernel */

/* GETBYTE: Get a character from inbuffer. Maintain inbufptr. */

char getbyte()
{
    char rtnbyte;

    rtnbyte = inbuffer[inbufptr];
    if (++inbufptr > BUFFERMAX) inbufptr = 0;
    return(rtnbyte);
}/* End getbyte */

/* PUTBYTE: Put a character in putbuffer. Maintain outbfptr. */

putbyte(inchar,putbuffr)
    char inchar;
    char putbuffr[];
{
    putbuffr[outbfptr] = inchar;
    if (++outbfptr > BUFFERMAX) outbfptr = 0;
} /* End putbyte */

/* LFTROT: Rotate a byte left, by number of bits. */

char lftrot(rotbyte,number)
    char rotbyte,number;
{
      asm           ldr.ixd b,ix+8; /* b <-- number of shifts */
      asm           ldr.ixd a,ix+6; /* a <-- byte to be shifted  */
      asm     lrot: rlcr a;         /* rotate the byte, while  */
      asm           djnze lrot;     /* decrementing b, jump on not zero  */
      asm           ld.ixdr ix+6,a; /* put rotated byte back */

  /* the return will put (ix+6) into the hl register pair */

return (rotbyte);

} /*End lftrot */

/* END DCT BENCHMARK PROGRAM ********************************************/
```

Tab 5

Whitesmith "C" Source

```
/*
 *      DCT BENCHMARK PROGRAM IN WHITESMITH'S C
 *
 *      Frank P. MacLachlan
 *      13-Oct-80
 *
 */

#include <std.h>        /* file containing standard definitions */

#define  ARRAYSIZE       125
#define  BITMASK         001
#define  BUFFERMAX       15
#define  INT1            300
#define  INT2            -150
#define  IOLOOPS         575
#define  IOPORT          0xff     /* adjust if conflict */
#define  MSGLENGTH       80
#define  NUMBERMSGS      200
#define  OPRACTIONLOOPS  100
#define  STARTCODE       'S'
#define  TESTBYTE        85
#define  TIMINGCONTROL   12

/*
 *      EXTERNAL VARIABLES:
 *      Due to an anomaly in the Whitesmith C compiler, all
 *      external variables must be given initial values.
 *      Therefore, external variables which require no initialization
 *      must be initialized to satisy the compiler.
 */
char    inbuf[BUFFERMAX+1]
                { 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
                  'S', 'A', 'A', 'A', 'A', 'A'};
int     array1[ARRAYSIZE+1] {0};
int     array2[ARRAYSIZE+1] {0};
int     inbufp {0};
int     outbufp {0};

struct  {
         char   charbuf[BUFFERMAX+1];
         int    intnumber;
} mixedtype { "", 0 };


/*********************************************\
*                                             *
*                Main Program                 *
*                                             *
\*********************************************/

_main()
        {
        register loopctr;       /* for speed */
        static int timinglooper;

        putfmt("Begin benchmark execution\n");
        for (loopctr = 0; loopctr <= ARRAYSIZE; ++loopctr)
                array2[loopctr] = loopctr;
        for (timinglooper = 0; timinglooper <= TIMINGCONTROL; ++timinglooper)
```

```
                kernal();
        for (loopctr = 0; loopctr <= BUFFERMAX; ++loopctr)
                putfmt("%a", mixedtype.charbuf[loopctr]);
        putfmt("\n%s\nEnd execution\n", mixedtype.intnumber);
        }


/**********************************************\
*                                              *
*               Support Functions              *
*                                              *
\**********************************************/

/*      get a character from inbuf, increment inbufp to
 *      next character.
 */
char getbyte()
        {
        static char c;

        c = inbuf[inbufp];
        if (++inbufp > BUFFERMAX)
                inbufp = 0;
        return (c);
        }


/*
 *      kernal is the main function in the benchmark program.
 *      It is declared VOID to indicate that no return value
 *      is expected.  VOID is defined in the file STD.H as
 *      being equivalent to int (16 bit integer).
 */
VOID kernal()
        {
        static char outbuf[BUFFERMAX+1];
        static char newchar;
        static int opaction;
        register loopctr, whilectr, i;  /* for speed */

        for (loopctr = 0; loopctr <= IOLOOPS; ++loopctr)
                out(IOPORT, ~TESTBYTE); /* standard library output routine */
        for (loopctr = 0; loopctr <= ARRAYSIZE; ++loopctr)
                array1[loopctr] = array2[ARRAYSIZE-loopctr];
        for (loopctr = 0; loopctr <= NUMBERMSGS; ++loopctr) {
                while ((newchar = getbyte()) != STARTCODE)
                        ;
                for (whilectr = 0; whilectr < MSGLENGTH; ++whilectr)
                        if (((newchar = getbyte()) >> 1) & 1)
                                putbyte(newchar, outbuf);
                        else
                                putbyte(newchar, outbuf);
        }
        opaction = 0;
        for (loopctr = 0; loopctr <= OPRACTIONLOOPS; ++loopctr) {
                switch(opaction) {

                        case 0:
                        case 3:
                        case 9:
                                mixedtype.charbuf[opaction] = (TESTBYTE >> 3);
                                break;

                        case 1:
                        case 4:
                        case 7:
                                mixedtype.charbuf[opaction] = lrot(TESTBYTE, 2)
                                break;

                        case 2:
                        case 6:
                                for (i = 0; i <= BUFFERMAX; ++i)
                                        mixedtype.charbuf[i] = inbuf[i];
                                break;

                        default:
                                mixedtype.intnumber = ((((INT1/INT2) * INT1)
                                  / INT2) * INT2) + INT1;
                        }
                if (++opaction > 9)
                        opaction = 0;
                }
        } /** end kernel **/


/*      put a byte into the next position in putbuf
 */
VOID putbyte(inchar, putbuf)
        char    inchar;
        char    putbuf[];
        {

        putbuf[outbufp] = inchar;
        if (++outbufp > BUFFERMAX)
                outbufp = 0;
        }


/*      rotate byte n left b bit positions
 *      (8 bit rotate)
 */
char lrot(n, b)
        char    n;
        char    b;
        {

        b &= 007;                       /* 0..7 */
        return((n << b) | (n >> 8-b));
        }
```

Tab 6

Microsoft FORTRAN-80 Source

```
C
C      PROGRAM DCT HOL BENCHMARK IN MICROSOFT FORTRAN-80 FOR CP/M
C      WRITTEN BY: CAPT B.F. BRADY, U.S.M.C
C
C***** VARIABLE DECLARATIONS FOR MAIN PROGRAM *****
      IMPLICIT LOGICAL(L-R)
C
      INTEGER OTBFPT,INBFPT,TIMLPR,INTNUM,ARRAY2
C
      LOGICAL*1 CHRBUF,INBUFF,BITMSK,TSTBYT
C
C***** DEFINE GLOBALS IN COMMON :
C
      COMMON //OTBFPT,INBUFF(16),INBFPT,CHRBUF(16),ARRAY2(126),
     *         INTNUM,BITMSK,TSTBYT
C
C***** PRESET DATA :
C
      DATA INBUFF/10*1HA,1HS,5*1HA/,BITMSK/1/,OTBFPT/0/,TSTBYT/1HU/
C
C***** BENCHMARK MAIN PROGRAM
C
C***** WRITE OUT THE START MESSAGE
      WRITE(5,1)
C***** INITIALIZE ARRAY2
      DO 102 TIMLPR=1,126
        ARRAY2(TIMLPR)=TIMLPR
C***** END INITIALIZE LOOP
102   CONTINUE
      DO 105 TIMLPR=0,12
C***** BEGIN MAIN TIMING LOOP
        CALL KERNEL
C***** END MAIN TIMING LOOP
105   CONTINUE
C***** WRITE OUT THE END MESSAGE
       WRITE(5,2)
C***** END OF MAIN PROGRAM
C***** FORMAT STATEMENTS :
1     FORMAT(' Begin BENCHMARK Execution.')
2     FORMAT(' End BENCHMARK Execution.')
      STOP
      END
C
C***** SUBROUTINES AND FUNCTIONS :
C
      FUNCTION GETBYT(DUMARG)
      LOGICAL*1 INBUFF,DUMARG
C***** DUMARG IS USED ONLY BECAUSE A PARAMETER IS
C***** REQUIRED FOR A FUNCTION CALL
C
      INTEGER OTBFPT,INBFPT,LOCPNT
C
      COMMON //OTBFPT,INBUFF(16),INBFPT
C
      LOCPNT=INBFPT
      INBFPT=INBFPT+1
      IF(INBFPT.GT.16) INBFPT=1
      GETBYT=INBUFF(LOCPNT)
      RETURN
C***** END GETBYT
      END
C
      SUBROUTINE PUTBYT(INCHAR,PUTBUF)
C
      LOGICAL*1 INCHAR,PUTBUF(16)
C
      INTEGER OTBFPT
C
      COMMON //OTBFPT
C
      PUTBUF(OTBFPT)=INCHAR
      OTBFPT=OTBFPT+1
      IF(OTBFPT.GT.16) OTBFPT=1
      RETURN
C***** END PUTBYT
      END
C
      SUBROUTINE KERNEL
      IMPLICIT LOGICAL(L-R)
C
      LOGICAL*1 NEWCHR,INBUFF,TSTBYT,BITMSK,CHRBUF,OTBUFF(16),START,
     *          BYTE
C
      INTEGER OTBFPT,INBFPT,INTNUM,LPCNTR,TIMLPR,WHLCNT,OPRACT,ARG2,
     *        ARRAY2,ARRAY1(126),INTGR1,INTGR2,TEMP
C
      COMMON //OTBFPT,INBUFF(16),INBFPT,CHRBUF(16),ARRAY2(126),
     *         INTNUM,BITMSK,TSTBYT
C
      DATA START/1HS/,INTGR1/300/,INTGR2/-150/
C
      DO 112 LPCNTR=0,575
         BYTE=.NOT.TSTBYT
         CALL OUT(BYTE,200)
112   CONTINUE
C
      DO 115 LPCNTR=1,126
         ARG2=127-LPCNTR
         ARRAY1(LPCNTR)=ARRAY2(ARG2)
115   CONTINUE
C
      DO 127 LPCNTR=0,200
C***** BEGIN MESSAGE LOOP
119     CONTINUE
        IF(GETBYT(BYTE).NE.START) GOTO 119
        WHLCNT=0
122   IF(WHLCNT.GT.80)GOTO 126
C***** BEGIN WHILE LOOP
          NEWCHR=GETBYT(BYTE)
          TEMP=NEWCHR
          IF(RTSHFT(TEMP,1).AND.BITMSK)GOTO 124
            CALL PUTBYT(NEWCHR,OTBUFF)
            GOTO 125
124     CONTINUE
            CALL PUTBYT(NEWCHR,OTBUFF)
125     CONTINUE
          WHLCNT=WHLCNT+1
C***** END WHILE LOOP
          GOTO 122
126   CONTINUE
C***** END MESSAGE LOOP
127   CONTINUE
      OPRACT=0
      DO 138 LPCNTR=0,100
C***** BEGIN OPERATOR ACTION LOOP
         INDEX=OPRACT+1
         GOTO(129,131,128,129,134,131,129,134,128),OPRACT
C
C***** CASE OF 0,3 OR 9
128       CHRBUF(INDEX)=RTSHFT(TSTBYT,3)
C***** END 0,3 OR 9
          GOTO 135
C
C***** CASE OF 1,4 OR 7
129       CHRBUF(INDEX)=LFTROL(TSTBYT,2).AND.BITMSK
C***** END 1,4 OR 7
          GOTO 135
C
C***** CASE OF 2 OR 6
131       DO 133  ARG2=1,16
             CHRBUF(ARG2)=INBUFF(ARG2)
133       CONTINUE
C***** END 2 OR 6
          GOTO 135
C
C***** OTHERWISE :
134       INTNUM=((((INTGR1/INTGR2)*INTGR1)/INTGR2)*INTGR2)+INTGR1
C***** END OTHERWISE
C
C***** END OF CASE SIMULATION
135      CONTINUE
         OPRACT=OPRACT+1
         IF(OPRACT.GT.9) OPRACT=0
138   CONTINUE
C***** END OPERATOR ACTION LOOP
C***** END KERNEL
      RETURN
      END
```

Tab 7

Cromemco RATFOR Source

```
#PROGRAM DCT HOL BENCHMARK in RATFOR for CP/M
#Written by: Capt B.F. BRADY, U.S.M.C
#
# INCLUDE RFGLBLS.RAT  # need this for TSW RatFor only
#
#DEFINE Constants :
#
DEFINE(arsize,126)
DEFINE(ioloop,575)
DEFINE(opaclp,100)
DEFINE(timctl,12)
DEFINE(ioport,200)
DEFINE(nmsgs,200)
DEFINE(tstbyt,85)
DEFINE(intgr1,300)
DEFINE(intgr2,(-150))
DEFINE(msglen,80)
DEFINE(strtcd,'S')
DEFINE(bufmax,16)
#
#Variable declarations for main program
#
INTEGER otbfpt,inbfpt,timlpr,intnum,array2
#
LOGICAL*1 chrbuf,inbuff,bitmsk,temp
#
#DEFINE globals in COMMON :
COMMON //otbfpt,inbuff(bufmax),inbfpt,chrbuf(bufmax),
            array2(arsize),temp,intnum,bitmsk
#
#PRESET DATA :
#
DATA inbuff/10*'A','S',5*'A'/,bitmsk/1/,otbfpt/0/
#
#Benchmark Main Program
#
    WRITE(5,1) #put start message on console
    # initialize array2 elements to equal their index
    FOR(inbfpt=1;inbfpt<=arsize;inbfpt=inbfpt+1)
         array2(inbfpt) = inbfpt
    # end initialize loop
    inbfpt = 0
    FOR(timlpr=1;timlpr<=timctl;timlpr=timlpr+1)
    {# BEGIN main timing loop
        CALL kernel
    }# END main timing loop
    WRITE(5,2)chrbuf,intnum
    # put out record values and end message on console
#FORMAT Statements:
 1 format(' begin benchmark execution')
 2 format(1x,16A1,I4,'  end execution')
STOP
END
#end of Main Program
#
```

```
#Subroutines and Functions :
#
FUNCTION setbyt(dumarg)
LOGICAL*1 inbuff,dumarg  #dumarg is used only because a parameter is
                         #required for a FUNCTION call
INTEGER   otbfpt,inbfpt,locpnt
#
COMMON //otbfpt,inbuff(bufmax),inbfpt
#
   locpnt = inbfpt
   inbfpt = inbfpt + 1
   IF(inbfpt .GT. bufmax)
      inbfpt = 1
   setbyt = inbuff(locpnt)
RETURN
END
#end setbyt
#
SUBROUTINE putbyt(inchar,putbuf)
#
LOGICAL*1 inchar,putbuf(bufmax)
#
INTEGER  otbfpt
#
COMMON //otbfpt
#
   putbuf(otbfpt) = inchar
   otbfpt = otbfpt + 1
   IF(otbfpt .GT. bufmax)
      otbfpt = 1
RETURN
END
#end putbyt
#
SUBROUTINE kernel
#
LOGICAL*1 newchr,inbuff,temp,bitmsk,chrbuf,otbuff(bufmax),start
#
INTEGER   otbfpt,inbfpt,intnum,timlpr,lpcntr,whlcnt,opract,arg2,
          array2,array1(arsize)
#
COMMON //otbfpt,inbuff(bufmax),inbfpt,chrbuf(bufmax),
         array2(arsize),temp,intnum,bitmsk
#
DATA     start/strtcd/
#
    FOR(lpcntr=1;lpcntr<=ioloop;lpcntr=lpcntr+1)
       {#BEGIN I/O Loop
        temp = (.NOT.tstbyt)
        CALL out(temp,ioport)
       }#END I/O Loop
    FOR(lpcntr=1;lpcntr<=arsize;lpcntr=lpcntr+1)
       {#BEGIN Array Loop
        arg2 = (arsize+1) - lpcntr
        array1(lpcntr) = array2(arg2)
   }#END Array Loop
FOR(lpcntr=1;lpcntr<=nmsgs;lpcntr=lpcntr+1)
   {#BEGIN Message Loop
    REPEAT
       newchr = getbyt(temp)  #use temp as a dummy here since
                              #at least one argument is needed for ca
    UNTIL(newchr .EQ. start)
    whlcnt = 0
    WHILE(whlcnt < msglen)
       {#BEGIN While Loop
        newchr = getbyt(temp)
        temp = newchr
        CALL rtshft(temp,1)
        IF(temp .AND. bitmsk)
           CALL putbyt(newchr,otbuff)
        ELSE
           CALL putbyt(newchr,otbuff)
        whlcnt = whlcnt + 1
       }#END While Loop
   }#END Message Loop
opract = 0
FOR(lpcntr=1;lpcntr<=opaclp;lpcntr=lpcntr+1)
   {#BEGIN Operator Action Loop
    ##################################################################
    ## simulate CASE statement with IF-ELSE-IF chain
    ##################################################################
    IF((opract.EQ.0).OR.(opract.EQ.3).OR.(opract.EQ.9))
       {#CASE of 0,3 or 9
        temp = tstbyt
        CALL rtshft(temp,3)
        chrbuf(opract) = temp
       }#End 0,3 or 9

    ELSE IF((opract.EQ.1).OR.(opract.EQ.4).OR.(opract.EQ.7))
       {#CASE of 1,4 or 7
        temp = tstbyt
        CALL lftrol(temp,2)
        temp = (temp .AND. bitmsk)
        chrbuf(opract) = temp
       }#End 1,4 or 7

    ELSE IF((opract.EQ.2).OR.(opract.EQ.6))
       {#CASE of 2 or 6
        FOR(arg2=1;arg2<=bufmax;arg2=arg2+1)
           chrbuf(arg2) = inbuff(arg2)
       }#End 2 or 6

    ELSE
       {#OTHERWISE :
        intnum=((((intgr1/intgr2)*intgr1)/intgr2)*intgr2)+intgr1
       }#End Otherwise
    #End of CASE Simulation
    opract = opract + 1
    IF(opract .GT. 9)
       opract = 0
       }#END Operator Action Loop
#end kernel
RETURN
END
```

Tab 8

PASCAL/MT Source

```
(*$L+*)
(* DCT Benchmark Program in Pascal/MT for CP/M *)
(*   written by : Capt B.F. BRADY, U.S.M.C.    *)
PROGRAM DCTBENCHMARK;
CONST
   arraysize  = 125;
   numbermsgs = 200;
   opracloops = 100;
   ioport     = 200;
   ioloops    = 575;
   timingcntrl=  12;
   testbyte   =  85;
   integer1   = 300;
   integer2   =-150;
   bitmask    =   1;
   msglength  =  80;
   startcode  = 'S';
   startbyte  =  10;
   buffermax  =  15;

TYPE
   buffertype = PACKED ARRAY[0..buffermax] of CHAR;
   mixedtype = RECORD
      charbuffer : buffertype;
      intnumber  : INTEGER
   END  (*RECORD mixedtype*);

VAR
   timinglooper,inbufptr,outbufptr : INTEGER;

   array1,array2 : ARRAY[0..arraysize] of INTEGER;
   inbuffer : buffertype;
   mixrec : mixedtype;

FUNCTION getbyte : CHAR;
   BEGIN
      getbyte := inbuffer[inbufptr];
      inbufptr := inbufptr + 1;
      IF inbufptr > buffermax THEN inbufptr := 0
   END  (*getbyte*);

PROCEDURE putbyte (VAR inchar : CHAR; VAR putbuffer : buffertype);
   BEGIN
      putbuffer[outbufptr] := inchar;
      outbufptr := outbufptr + 1;
      IF outbufptr > buffermax THEN outbufptr := 0
   END  (*putbyte*);


PROCEDURE LFTROL(VAR temp : INTEGER; bits :INTEGER);
BEGIN
   INLINE(          "LDA / bits/
                    "MOV  B,A /
                    "LDA / temp/
         [LROT]/    "RLC  /
```

```
                  "DCR  B /
                  "JNZ / LROT/
                  "STA / temp);
END;

PROCEDURE kernel;

   VAR outbuffer : buffertype;
       loopcounter,whilecounter,opraction,temp : INTEGER;
       newchar : CHAR;

   BEGIN
      FOR loopcounter := 0 TO ioloops DO
          BEGIN
             temp := testbyte;
             (* use in-line direct code to complement temp and put *)
             (* it out the specified port *)
             INLINE( "LDA /temp/
                     "CMA /
                     "OUT /ioport);
      END  (*FOR*);

      FOR loopcounter := 0 TO arraysize DO
         array1[loopcounter] := array2[arraysize - loopcounter];

      FOR loopcounter := 0 TO numbermsgs DO
         BEGIN
            REPEAT
               newchar := getbyte
            UNTIL newchar = startcode;
            whilecounter := 0;

            WHILE whilecounter < msglength DO
               BEGIN
                  newchar := getbyte;
                  temp := SHR(newchar,1);
                 IF (ODD(temp) AND ODD(bitmask)) THEN
                     putbyte(newchar,outbuffer)
                  ELSE
                     putbyte(newchar,outbuffer);
                  whilecounter := whilecounter + 1;
               END   (* WHILE*);
            END  (*FOR*);

   opraction := 0;
   FOR loopcounter := 0 TO opracloops DO
      BEGIN
         temp := testbyte;
         CASE opraction OF

            0,3,9  : mixrec.charbuffer[opraction] := CHR(SHR(temp,3));

            1,4,7  : BEGIN
                        LFTROL(temp,2);
                        mixrec.charbuffer[opraction] :=
                              CHR(ODD(temp) AND ODD(bitmask))
                     END;

            2.6    : mixrec.charbuffer := inbuffer;

         ELSE
            mixrec.intnumber := ((((integer1 DIV integer2)*integer1)
                                    DIV integer2)*integer2) + integer1
         END   (*CASE*);

         opraction := opraction + 1;
         IF opraction > 9 then opraction := 0
      END  (*FOR*);
   END  (*kernel*);

BEGIN (* MAIN PROGRAM EXECUTION *)
   WRITELN(' Begin Benchmark Execution');

   FOR timinglooper := 0 TO arraysize DO
       array2[timinglooper] := timinglooper;

   FOR timinglooper := 0 TO buffermax DO
       inbuffer[timinglooper] := 'A';

   inbuffer[startbyte] := startcode;
   inbufptr := 0;
   outbufptr := 0;
   FOR timinglooper := 0 TO timingcntrl DO
       kernel;
   writeln(mixrec.charbuffer,mixrec.intnumber);
   WRITELN(' End Benchmark Execution')
END.
```

Tab 8
PASCAL/MT Source

```
(*$I  intersperse pascal source in asmbl source   *)

(* DCT HOL BENCHMARK in Pascal/Z from Ithaca Intersystems *)
(* Written by: Capt B.F. BRADY, U.S.M.C. *)

PROGRAM DCTBENCHMARK;

CONST
   arraysize   = 125;
   numbermsgs  = 200;
   opracloops  = 100;
   ioport      = 200;
   ioloops     = 575;
   timingcntrl=   2;
   testbyte    =  85;
   integer1    = 300;
   integer2    =-150;
   bitmask     =   1;
   msglength   =  80;
   startbyte   =  10;
   buffermax   =  15;

TYPE
   buffertype = ARRAY[0..buffermax] of CHAR;
   mixedtype = RECORD
      charbuffer : buffertype;
      intnumber  : INTEGER
   END  (*RECORD mixedtype*);

VAR
   timinglooper : INTEGER;
   inbufptr,outbufptr : INTEGER;
   array1,array2 : ARRAY[0..arraysize] of INTEGER;
   inbuffer : buffertype;
   mixrec : mixedtype;

FUNCTION setbyte : CHAR;

   VAR
      localpnt : INTEGER;

   BEGIN
      localpnt := inbufptr;
      inbufptr := inbufptr + 1;
      IF inbufptr > buffermax THEN inbufptr := 0;
      setbyte := inbuffer[localpnt]
   END  (*setbyte*);

PROCEDURE putbyte (VAR inchar : CHAR; VAR putbuffer : buffertype);

   BEGIN
      putbuffer[outbufptr] := inchar;
      outbufptr := outbufptr + 1;
      IF outbufptr > buffermax THEN outbufptr := 0
   END  (*putbyte*);
```

```
PROCEDURE ANDCHR(VAR temp : INTEGER; mask : INTEGER);EXTERNAL;

PROCEDURE OUTPUT(VAR temp : INTEGER; port : INTEGER);EXTERNAL;

PROCEDURE LFTROL(VAR temp : INTEGER; bits : INTEGER);EXTERNAL;

PROCEDURE RTSHFT(VAR temp : INTEGER; bits : INTEGER);EXTERNAL;

PROCEDURE kernel;

  VAR
    outbuffer : buffertype;
    loopcounter,whilecounter : INTEGER;
    opraction,temp : INTEGER;
    newchar : CHAR;

  BEGIN
writeln('kernel');
writeln('ioloops');
    FOR loopcounter := 0 TO ioloops DO
      BEGIN
        temp := testbyte;
        OUTPUT(temp,ioport)
      END  (*FOR*);

writeln('arrayloops');
    FOR loopcounter := 0 TO arraysize DO
      array1[loopcounter] := array2[arraysize - loopcounter];

writeln('msgloops');
    FOR loopcounter := 0 TO numbermsgs DO
      BEGIN
        REPEAT
          newchar := getbyte
        UNTIL newchar = 'S';

        whilecounter := 0;
        WHILE whilecounter < msglength DO
          BEGIN
            newchar := getbyte;
            temp := ord(newchar);
            RTSHFT(temp,1);
            ANDCHR(temp,bitmask);
            IF temp = 1  THEN
              putbyte(newchar,outbuffer)
            ELSE
              putbyte(newchar,outbuffer);
            whilecounter := whilecounter + 1
          END   (* WHILE*)
        END  (*FOR*);

  opraction := 0;
writeln('oprloops');
  FOR loopcounter := 0 TO opracloops DO
```

```
      BEGIN
         temp := testbyte;

(*$J9 Compiler option to create CASE Jump Table for 0..9 *)
         CASE opraction OF

            0,3,9  : BEGIN
                        RTSHFT(temp,3);
                        mixrec.charbuffer[opraction] := chr(temp)
                     END;

            1,4,7  : BEGIN
                        LFTROL(temp,2);
                        ANDCHR(temp,bitmask);
                        mixrec.charbuffer[opraction] := chr(temp)
                     END;

            2,6    : mixrec.charbuffer := inbuffer;

         ELSE:
            mixrec.intnumber := ((((integer1 DIV integer2)*integer1)
                                    DIV integer2)*integer2) + integer1
         END   (*CASE*);

         opraction := opraction + 1;
         IF opraction > 9 then opraction := 0
      END  (*FOR*)
   END  (*kernel*);

BEGIN (* MAIN PROGRAM EXECUTION *)
   WRITELN(' Begin Benchmark Execution');

   FOR timinglooper := 0 TO arraysize DO
       array2[timinglooper] := timinglooper;

   FOR timinglooper := 0 TO buffermax DO
       inbuffer[timinglooper] := 'A';

   inbuffer[startbyte] := 'S';
   inbufptr := 0;
   outbufptr := 0;
   FOR timinglooper := 0 TO timingcntrl DO
      kernel;
   WRITELN(mixrec.charbuffer,mixrec.intnumber);
   WRITELN(chr(7),' End Benchmark Execution')
END.
```

Tab 10

PLI-80 Source (Version 1)

```
/* BENCHMARK PROGRAM IN PLI/80 FOR CP/M */
/* Written by: Capt B.F. BRADY, U.S.M.C */

BENCHMARK_PLI_80:
  PROCEDURE OPTIONS(MAIN);
    %REPLACE /* DEFINE CONSTANTS */
      opr_action_loops BY  100,
      timing_control   BY    1,
      array_size       BY  125,
      io_loops         BY  575,
      io_port          BY  200,
      number_msgs      BY  200,
      test_byte        BY '55'B4,
      buffer_max       BY   15,
      integer1         BY  300,
      integer2         BY  150,
      bit_mask         BY '1'B,
      msg_length       BY   80,
      start_byte       BY   10,
      start_code       BY  'S';
/*
DEFINE EXTERNAL PROCEDURE ENTRIES
*/
  DCL
    OUTPUT     ENTRY(BIT(8),BIT(8)),
    RTSHFT     ENTRY(BIT(8),FIXED(7)),
    LFTROT     ENTRY(BIT(8),FIXED(7));

/*
DEFINE VARIABLES FOR MAIN PROGRAM :
*/

  DCL
    1 mixed_type STATIC,
      2 character_buffer(0:buffer_max) CHAR(1),
      2 integer_number FIXED BINARY;
  DCL
    input_buffer(0:buffer_max) CHAR(1) INITIAL((10)'A','S',(5)'A') STATIC,
    out_buffer_pointer FIXED BINARY(7) STATIC,
    in_buffer_pointer FIXED BINARY(7) STATIC,
    timing_loop_counter FIXED BINARY STATIC,
    i FIXED BINARY(7) STATIC,
    array_2(array_size) FIXED BINARY STATIC;

/*
PROCEDURE AND FUNCTION DEFINITIONS:
*/

  PUT_BYTE:
    PROCEDURE(in_character,put_buffer);
    DCL
      in_character CHAR(1),
      put_buffer(0:buffer_max) CHAR(1);
    BEGIN;
      put_buffer(out_buffer_pointer) = in_character;
      out_buffer_pointer = out_buffer_pointer + 1;
      IF out_buffer_pointer > buffer_max THEN
        out_buffer_pointer = 0;
    END;
  END PUT_BYTE;

  GET_BYTE:
    PROCEDURE RETURNS(CHAR(1)); /* A FUNCTION DEFINITION */
    DCL
      local_pointer FIXED BINARY(7);
    BEGIN;
      /* SAVE INDEX IN local_pointer FOR THE RETURN */
      local_pointer =in_buffer_pointer;
      in_buffer_pointer = in_buffer_pointer + 1;
      IF in_buffer_pointer > buffer_max THEN
        in_buffer_pointer = 0;
    END;
    RETURN(input_buffer(local_pointer));
  END GET_BYTE; /* END FUNCTION GET_BYTE */

KERNEL:
  PROCEDURE;
    DCL
      out_buffer(0:buffer_max) CHAR(1) STATIC,
      loop_counter FIXED BINARY STATIC,
      array_1(array_size) FIXED BINARY STATIC,
      new_character CHAR(1) STATIC,
      temp FIXED BINARY(7) STATIC,
      operator_action FIXED BINARY(7) STATIC,
      while_counter FIXED BINARY(7) STATIC;
    BEGIN;
      DO loop_counter = 0 TO io_loops;
        CALL output(io_port,^test_byte);
      END; /* I/O Loop  */
      DO loop_counter = 0 TO array_size;
        array_1(loop_counter)=array_2(array_size - loop_counter);
      END; /* Array loop  */
      DO loop_counter = 0 TO number_msgs;
        DO WHILE(get_byte() ^= start_code);
        END; /* DO WHILE  */
        while_counter = 0;
        DO WHILE(while_counter < msg_length);
          new_character = get_byte();
          temp = UNSPEC(new_character);
          CALL rtshft(1,temp);
          IF (UNSPEC(temp) & bit_mask) THEN
            CALL put_byte(new_character,out_buffer);
          ELSE
            CALL put_byte(new_character,out_buffer);
          while_counter = while_counter + 1;
        END;  /* WHILE Loop  */
      END;  /* Message Loop  */
      operator_action = 0;
      DO loop_counter = 0 TO opr_action_loops;
  put skip list(operator_action,mixed_type.character_buffer(operator_actior
          GOTO CASE(operator_action);

      CASE(0): ;
      CASE(3): ;
      CASE(9):
                 temp = test_byte;
                 CALL rtshft(3,temp);
                 UNSPEC(mixed_type.character_buffer(operator_action)) = UNSPEC(te
                 GOTO END_CASE;

      CASE(1): ;
      CASE(4): ;
      CASE(7):
                 temp = test_byte;
                 CALL lftrot(2,temp);
                 UNSPEC(mixed_type.character_buffer(operator_action)) =
                 (UNSPEC(temp) & bit_mask);
                 GOTO END_CASE;

      CASE(2): ;
      CASE(6):
                 mixed_type.character_buffer = input_buffer;
                 GOTO END_CASE;

      /* OTHERWISE */
      CASE(5): ;
      CASE(8):
                 mixed_type.integer_number =
             ((((integer1/(-integer2))*integer1)/(-integer2))*(-integer2))+intege

      END_CASE:
        put skip list(operator_action,mixed_type.character_buffer(operator_actior
                 operator_action = operator_action + 1;
                 IF operator_action > 9 THEN operator_action = 0;
           END; /* Operator Action Loop */
         END;
      END KERNEL;

      /*********************************/
      /* Begin main program execution */
      /*********************************/

        BEGIN;
          PUT SKIP LIST(' Begin Benchmark Execution');
          input_buffer(start_byte) = start_code;
          DO timing_loop_counter = 0 TO array_size;
            array_2(timing_loop_counter) = timing_loop_counter;
          END; /* initialize array_2  */
          DO timing_loop_counter = 0 TO timing_control;
            CALL kernel;
          END; /* main timing loop */
          PUT SKIP;
          DO i=1 TO buffer_max;
          PUT LIST(mixed_type.character_buffer(i));
          END;
```

```
    PUT LIST(mixed_type.integer_number);
    PUT SKIP LIST(' End Execution');
  END;
END BENCHMARK_PLI_80; /* Main Program */
```

Tab 11

PLI-80 Source (Version 2)

```
FILENAME: BMPLIS.PLI     17-NOV-8Ø

BMARK: PROCEDURE OPTIONS (MAIN);

/**********************************************\
*                                              *
*       DCT BENCHMARK PROGRAM IN PL/I-8Ø       *
*                                              *
*       WRITTEN BY:   F. P. MACLACHLAN         *
*       DATE:         14-OCT-8Ø                *
*                                              *
\**********************************************/

   %REPLACE
      ARRAY_SIZE         BY 125,
      BITMASK            BY '1'B,
      BUFFER_MAX         BY 15,
      INTEGER1           BY 3ØØ,
/*    INTEGER2           BY (-15Ø),       WON'T ACCEPT (-15Ø)!!
*/
      IO_LOOPS           BY 575,
      IO_PORT            BY 2ØØ,          /* ADJUST */
      MSG_LENGTH         BY 8Ø,
      NUMBER_MSGS        BY 2ØØ,
      OPR_ACTION_LOOPS   BY 1ØØ,
      START_CODE         BY 'S',
      TEST_BYTE          BY '55'B4,       /* 85 DECIMAL */
      TIMING_CONTROL     BY 12;

   DCL
      SHIFTR             ENTRY (BIT(8), FIXED(7)) RETURNS (BIT(8)),
      ROTATL             ENTRY (BIT(8), FIXED(7)) RETURNS (BIT(8)),
      OUTPUT             ENTRY (FIXED(7), BIT(8));

   DCL
      NEW_CHAR           CHAR,
      OUT_BUFFER(Ø:15)   CHAR STATIC,     /* NOTE: COULD USE CHAR(16) */

      ARRAY1(Ø:ARRAY_SIZE) FIXED BINARY,
      ARRAY2(Ø:ARRAY_SIZE) FIXED BINARY,
      IN_BUFFER(Ø:15) CHAR STATIC INITIAL
         ('A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
          'S', 'A', 'A', 'A', 'A', 'A'),
      IN_BUFFER_PTR      FIXED BINARY STATIC INITIAL (Ø),
      LOOP_COUNTER       FIXED BINARY,
      OPR_ACTION         FIXED BINARY,
      OUT_BUFFER_PTR     FIXED BINARY STATIC INITIAL (Ø),
      TIMING_LOOPER      FIXED BINARY,
      WHILE_COUNTER      FIXED BINARY,

      1 MIXED_TYPE STATIC,
         2 CHAR_BUFFER(Ø:15)    CHAR,
         2 INT_NUMBER           FIXED BINARY;

GET_BYTE:
   PROCEDURE RETURNS (CHAR);
   DCL
      C CHAR;

   C = IN_BUFFER(IN_BUFFER_PTR);
   IN_BUFFER_PTR = IN_BUFFER_PTR + 1;
   IF IN_BUFFER_PTR > BUFFER_MAX THEN
      IN_BUFFER_PTR = 0;
   RETURN (C);
   END GET_BYTE;


PUT_BYTE:
   PROCEDURE (IN_CHAR, PUT_BUFFER);
   DCL
      IN_CHAR CHAR,
      PUT_BUFFER(0:BUFFER_MAX) CHAR;

   PUT_BUFFER(OUT_BUFFER_PTR) = IN_CHAR;
   OUT_BUFFER_PTR = OUT_BUFFER_PTR + 1;
   IF OUT_BUFFER_PTR > BUFFER_MAX THEN
      OUT_BUFFER_PTR = 0;
   END PUT_BYTE;


KERNAL:
   PROC;
   DCL
      OUT_BUFFER(0:BUFFER_MAX) CHAR;

   DO LOOP_COUNTER = 0 TO IO_LOOPS;
      CALL OUTPUT(IO_PORT, ^TEST_BYTE);
      END;
   DO LOOP_COUNTER = 0 TO ARRAY_SIZE;
      ARRAY1(LOOP_COUNTER) = ARRAY2(ARRAY_SIZE-LOOP_COUNTER);
      END;
   DO LOOP_COUNTER = 0 TO NUMBER_MSGS;
      NEW_CHAR = GET_BYTE();
      DO WHILE (NEW_CHAR ^= START_CODE);
         NEW_CHAR = GET_BYTE();
         END;
      WHILE_COUNTER = 0;
      DO WHILE (WHILE_COUNTER < MSG_LENGTH);
         NEW_CHAR = GET_BYTE();
         IF SHIFTR(UNSPEC(NEW_CHAR), 1) & '1'B THEN
            CALL PUT_BYTE(NEW_CHAR, OUT_BUFFER);
         ELSE
            CALL PUT_BYTE(NEW_CHAR, OUT_BUFFER);
         WHILE_COUNTER = WHILE_COUNTER + 1;
         END; /* DO WHILE */
      END; /* DO LOOP_COUNTER */

   OPR_ACTION = 0;
   DO LOOP_COUNTER = 0 TO OPR_ACTION_LOOPS;
      GOTO CASE(OPR_ACTION);
```

FILENAME: BMPLIS.PLI   17-NOV-82

```
CASE(Ø):;
CASE(3):;
CASE(9):;
         UNSPEC(MIXED_TYPE.CHAR_BUFFER(OPR_ACTION)) =
            SHIFTR(TEST_BYTE, 3);
         GOTO END_CASE;

CASE(1):;
CASE(4):;
CASE(7):;
         UNSPEC(MIXED_TYPE.CHAR_BUFFER(OPR_ACTION)) =
            ROTATL(TEST_BYTE, 2) & BITMASK;
         GOTO END_CASE;

CASE(2):;
CASE(6):;
         MIXED_TYPE.CHAR_BUFFER = IN_BUFFER;
         GOTO END_CASE;

CASE(5):;
CASE(8):;
         MIXED_TYPE.INT_NUMBER =
            ((((INTEGER1 / (-15Ø)) * INTEGER1) / (-15Ø)) * (-15Ø)) + INTEGER1;
END_CASE:;
      OPR_ACTION = OPR_ACTION + 1;
      IF OPR_ACTION > 9 THEN
         OPR_ACTION = Ø;
      END; /* DO LOOP_COUNTER = Ø TO OPR_ACTION_LOOPS */

   END KERNAL;


/***************************\
*   START OF MAIN PROGRAM   *
\***************************/
   DCL
      I FIXED BINARY;


   PUT SKIP LIST ('BEGIN BENCHMARK EXECUTION');
   DO I = Ø TO BUFFER_MAX;
      ARRAY2(I) = I;
      END;
   DO TIMING_LOOPER = Ø TO TIMING_CONTROL;
      CALL KERNAL();
      END;
   DO I = Ø TO 15;
      PUT LIST (MIXED_TYPE.CHAR_BUFFER(I));
      END;
   PUT LIST
       (MIXED_TYPE.INT_NUMBER,
        'END EXECUTION');
   END BMARK;
```

Tab 12

PLMX Source

.Benchmark Source

```
PL/MX COMPILER VERSION 2.4
COPYRIGHT (C) 1980, SYSTEMS CONSULTANTS, INC.
END OF FAST COMPILATION
000 ERROR(S) DETECTED
/**********************************************\
*                                              *
*       DCT BENCHMARK - PL/M VERSION           *
*                                              *
\**********************************************/

bmark: do;
   declare
      equ         literally      'literally',
      ARRAY$SIZE           equ     '126',   /* 0..125 */
      BITMASK              equ     '01h',
      BUFFER$SIZE          equ     '16',    /* 0..15 */
      CR                   equ     '0Dh',
      INT1                 equ     '300',
      INT2                 equ     '-150',
      IO$LOOPS             equ     '575',
      IO$PORT              equ     '200',   /* adjust */
      LF                   equ     '0Ah',
      MSG$LENGTH           equ     '80',
      NUMBER$MSGS          equ     '80',
      OPR$ACTION$LOOPS     equ     '100',
      START$CODE           equ     '053h',  /* ascii big S */
      TEST$BYTE            equ     '85',
      TIMING$CONTROL       equ     '12';

   declare
      array1(ARRAY$SIZE) address,
      array2(ARRAY$SIZE) address,
      i address,
      in$buffer(BUFFER$SIZE) byte initial ('AAAAAAAAAASAAAAA'),
      in$buffer$ptr address initial (0),
      out$buffer$ptr address initial (0),
      timing$looper address;

   declare
      mixed$type structure (
         char$buffer(BUFFER$SIZE) byte,
         int$number address
         );

nmout: procedure (value, base, lc, buf, width) external;
   declare
      (value, buf) address,
      (base, lc, width) byte;
   end nmout;


/*
 * signed 16 bit divide (d2 /d1)
 */
divid: procedure (d1, d2) address external;
   declare
      (d1, d2) address;
   end divid;
 * print character string until $ encountered
 */
pr$buf: procedure (strp) external;
   declare
      strp address;
   end pr$buf;

/*
 * print character at console
 */
wr$con: procedure (ch) external;
   declare
      ch byte;
   end wr$con;

/*
 * print address variable at console with one leading space
 */
putdec: procedure (num);
   declare
      num address,
      buf(7) byte;               /* scratch buffer for conversion */

   call nmout(num, 10, ' ', .buf, 6);
   buf(6) = '$';
   call pr$buf(.buf);
   end putdec;


/*
 * get the next byte from in$buffer, increment in$buffer$ptr
 */
get$byte: procedure byte;
   declare
      c byte;

   c = in$buffer(in$buffer$ptr);
   in$buffer$ptr = in$buffer$ptr + 1;
   if in$buffer$ptr > last(in$buffer) then
      in$buffer$ptr = 0;
   return c;
   end get$byte;


/*
 * put character into buffer, increment out$buffer$ptr
 */
put$byte: procedure (in$char, pb);
   declare
      in$char byte,
      (pb, pc) address,
      put$char based pc byte;

   pc = pb + out$buffer$ptr;     /* compute ptr to next cell in buffer */
   put$char = in$char;
   out$buffer$ptr = out$buffer$ptr + 1;
   if out$buffer$ptr > BUFFER$SIZE - 1 then
      out$buffer$ptr = 0;
   end put$byte;
```

```
/*
 * kernel is the main procedure in the benchmark program.
 */
kernel: procedure;
```

```
      (temp1, temp2) address,
      i address,
      loop$counter address,
      new$char byte,
      opr$action address,
      out$buffer(BUFFER$SIZE) byte,
      while$counter address;

   do loop$counter = 0 to IO$LOOPS;
      output(IO$PORT) = not TEST$BYTE;
      end;
   do loop$counter = 0 to last(array1);
      array1(loop$counter) = array2(ARRAY$SIZE-loop$counter);
      end;
   do loop$counter = 0 to NUMBER$MSGS;
      do while ((new$char := get$byte) <> START$CODE);
         end;
      while$counter = 0;
      do while (while$counter < MSG$LENGTH);
         if shr((new$char := get$byte), 1) then /* tests only bit 0 */
            call put$byte(new$char, .out$buffer);
         else
            call put$byte(new$char, .out$buffer);
         while$counter = while$counter + 1;
         end; /* do while */
      end; /* do loop$counter */

   opr$action = 0;
   do loop$counter = 0 to OPR$ACTION$LOOPS;
      if opr$action = 0 or opr$action = 3 or opr$action = 9 then
         mixed$type.char$buffer(opr$action) =
            shr(TEST$BYTE, 3);
      else if opr$action = 1 or opr$action = 4 or opr$action = 7 then
         mixed$type.char$buffer(opr$action) =
            rol(TEST$BYTE, 2) and BITMASK;
      else if opr$action = 2 or opr$action = 6 then
         do i = 0 to last(mixed$type.char$buffer);
            mixed$type.char$buffer(i) = in$buffer(i);
            end;
      else do;  /* default */
         temp1 = divid(INT2, INT1) * INT1;
         temp2 = divid(INT2, temp1);
         mixed$type.int$number = temp2 * INT2 + INT1;
         end;
      opr$action = opr$action + 1;
      if opr$action > 9 then
         opr$action = 0;
      end; /* do loop$counter = 0 to OPR$ACTION$LOOPS */

   end kernel;

/***************************\
*   start of main program   *
\***************************/

   call pr$buf(.('Begin benchmark execution', CR, LF, '$'));
   do i = 0 to last(array2);
      array2(i) = i;
      end;
   do timing$looper = 0 to TIMING$CONTROL;
      call kernel;
      end;
   do i = 0 to BUFFER$SIZE - 1;
      call wr$con(mixed$type.char$buffer(i));
      end;
```

```
call pr$buf(.('End Exacution', CR, LF, '$'));
end bmark;
```

Tab 13

PLZ Source

```
! THIS IS THE MCTSSA DCT BENCHMARK IN PLZ. CREATED AT ANTHEM ON 24SEP80. !
! DATA GATHERED ON 2OCT80. USED ZDS 1/40 MDS. PLZ REVISION H.  !

DCTBENCHMARK MODULE
  CONSTANT
    ARRAYSIZE := 125;  IOLOOPS := 575; OPRACTIONLOOPS := 100;
    TIMINGCONTROL := 12;  IOPORT :=01; NUMBERMSGS := 200; TESTBYTE := 35;
    INTEGER1 := 300; INTEGER2 := -150; BITMASK := 1; MSGLENGTH := 30
    BUFFERMAX := 15; STARTCODE := '$'; CONOUT := 2;


  TYPE

  CHAR BYTE; ! CHAR IS OF TYPE BYTE !
  BUFFERTYPE  ARRAY[BUFFERMAX + 1: CHAR];
  CHARBUFPTR  ↑BUFFERTYPE;

 EXTERNAL


  PUTSTRING PROCEDURE(UNIT CHAR, PTR ↑CHAR);
  PUTCHARS PROCEDURE(UNIT CHAR, PTR ↑CHAR, LEN WORD);
  PUTINTEGER PROCEDURE(UNIT CHAR, INT INTEGER);
  OUTPUT PROCEDURE(FIRST : CHAR , SECOND : CHAR);
  SHIFTR PROCEDURE(THIRD : CHAR , FOURTH : CHAR)
        RETURNS (RSLT1 : CHAR);
  ROTATL PROCEDURE(FIFTH : CHAR , SIXTH : CHAR)
        RETURNS (RSLT2 : CHAR);
  DIVID PROCEDURE(DIVISOR : INTEGER , DIVIDEND : INTEGER)
        RETURNS(QUOTIENT : INTEGER);


 ! BEGIN GLOBAL (TO THIS MODULE) DATA DECLARATIONS !

 INTERNAL

  TIMINGLOOPER, LOOPCOUNTER, WHILECOUNTER, OPRACTION,
  INBUFFERPTR, OUTBUFFERPTR, TEMP1 , TEMP2 : INTEGER;

  ARRAY1, ARRAY2 : ARRAY[ARRAYSIZE + 1: INTEGER];

  INBUFFER : BUFFERTYPE;

  NEWCHAR : CHAR;

  MIXEDTYPE : RECORD[
    CHARBUFFER : BUFFERTYPE;
    INTNUMBER : INTEGER; ]   ! END RECORD MIXEDTYPE !

 ! END DATA DECLARATIONS !
 ! BEGIN INTERNAL PROCEDURE DECLARATIONS/DEFINITIONS !


 GETBYTE PROCEDURE RETURNS(RETURNBYTE : CHAR);
  ENTRY
  RETURNBYTE := INBUFFER[INBUFFERPTR];
  INBUFFERPTR += 1;
  IF INBUFFERPTR > BUFFERMAX THEN INBUFFERPTR := 0 FI;
  END GETBYTE;


 PUTBYTE PROCEDURE (INCHAR : CHAR, CHARPTR : CHARBUFPTR);
 ! NOTE THAT CHARPTR POINTS TO TYPE BUFFERTYPE. IT SHOULD BE PASSED
 THE ADDRESS OF THE BEGINNING OF THE ARRAY UPON WHICH IT WILL OPERATE.  !

  ENTRY
  CHARPTR↑[OUTBUFFERPTR] := INCHAR;
  OUTBUFFERPTR += 1;
  IF OUTBUFFERPTR > BUFFERMAX THEN OUTBUFFERPTR := 0 FI;
  END PUTBYTE;


 KERNEL PROCEDURE;

  LOCAL
    I : INTEGER;    ! LOCAL COUNTER !
    OUTBUFFER : BUFFERTYPE;
  ! BOTH I AND OUTBUFFER ARE AUTOMATIC/DYNAMICALLY ALLOCATED !
```

```
ENTRY

LOOPCOUNTER := 0;
DO
        IF LOOPCOUNTER > IOLOOPS THEN EXIT FI;
        OUTPUT(NOT(TESTBYTE), IOPORT);
        LOOPCOUNTER +=1;
OD

LOOPCOUNTER := 0;

DO
        IF LOOPCOUNTER > ARRAYSIZE THEN EXIT FI;
        ARRAY1[LOOPCOUNTER] := ARRAY2[ARRAYSIZE - LOOPCOUNTER];
        LOOPCOUNTER += 1;
OD


LOOPCOUNTER := 0;

DO
        IF LOOPCOUNTER > NUMBERMSGS THEN EXIT FI;
        DO
                NEWCHAR := GETBYTE;
                IF NEWCHAR = STARTCODE THEN EXIT FI;
        OD
        WHILECOUNTER := 0;

        DO
                IF WHILECOUNTER >= MSGLENGTH THEN EXIT FI;
                NEWCHAR := GETBYTE;

                IF (SHIFTR(NEWCHAR, 1) AND BITMASK) = 1 THEN
                  PUTBYTE(NEWCHAR, #OUTBUFFER);
                ELSE
                  PUTBYTE(NEWCHAR, #OUTBUFFER);
                FI;

                WHILECOUNTER += 1;

        OD  ! END OF PLZ FORM OF DO-WHILE LOOP !

        LOOPCOUNTER += 1;
OD  ! END OF FOR LOOPCOUNTER FROM 0 TO NUMBERMSGS LOOP  !

OPRACTION := 0;
LOOPCOUNTER := 0;

DO
        IF LOOPCOUNTER > OPRACTIONLOOPS THEN EXIT FI;

        IF OPRACTION
                CASE 0,3,9  THEN    MIXEDTYPE.CHARBUFFER[OPRACTION] :=
                                   SHIFTR(TESTBYTE, 3);

                CASE 1,4,7  THEN    MIXEDTYPE.CHARBUFFER[OPRACTION] :=
                                   ROTATL(TESTBYTE, 2) AND BITMASK;

                CASE 2,6    THEN
                                               I := 0;
                                               DO
                                                 IF I > BUFFERMAX THEN EXIT FI;
                                                 MIXEDTYPE.CHARBUFFER[I] :=
                                                    INBUFFER[I];
                                                 I += 1;
                                               OD
                ELSE
                                    TEMP1 := DIVID(INTEGER2 , INTEGER1) * INTEGER1;
                                    TEMP2 := DIVID(INTEGER2 , TEMP1);
                                    MIXEDTYPE.INTNUMBER :=
                                               TEMP2 * INTEGER2 + INTEGER1;
! NOTE:  ALTHOUGH NOT CLEAR FROM THE DOCUMENTATION, THE DIVIDE SHOWN BELOW WILL
NOT WORK SINCE PLZ HANDLES NEGATIVE NUMBERS INTERNALLY AS THOUGH THEY WERE
LARGE POSITIVE NUMBERS. FOR EXAMPLE, 4/(-2) IS EQUAL TO ZERO.  *******  !
                                    ! MIXEDTYPE.INTNUMBER := ((((INTEGER1/INTEGER2)
                                      * INTEGER1)/INTEGER2)*INTEGER2) + INTEGER1; !
        FI;  ! END CASE !
        OPRACTION += 1;
        LOOPCOUNTER +=1;
```

```
            IF OPRACTION > 9 THEN OPRACTION := 0 FI;
      OD  ! END FOR LOOPCOUNTER FROM 0 TO OPRACTIONLOOPS !
   END KERNEL;
   GLOBAL : ******************* GLOBAL DEFINITION OF MAIN PROGRAM  **************
   MAIN PROCEDURE
      ENTRY
      PUTSTRING (CONOUT, #'BEGIN BENCHMARK EXECUTION.%R/');

      LOOPCOUNTER := 0;
      DO
            IF LOOPCOUNTER > ARRAYSIZE THEN EXIT FI;
            ARRAY2[LOOPCOUNTER] := LOOPCOUNTER;
            LOOPCOUNTER += 1;
      OD

      INBUFFERPTR := 0;
      DO
            IF INBUFFERPTR > BUFFERMAX THEN EXIT FI;
            INBUFFER[INBUFFERPTR] := 'A';
            INBUFFERPTR += 1;
      OD

      INBUFFER[10] := STARTCODE;

      INBUFFERPTR := 0;
      OUTBUFFERPTR := 0;

      TIMINGLOOPER := 0;
      DO
            IF TIMINGLOOPER > TIMINGCONTROL THEN EXIT FI;
            KERNEL;
            TIMINGLOOPER += 1;
      OD

      PUTCHARS(CONOUT, %MIXEDTYPE.CHARBUFFER[0], BUFFERMAX + 1);
      PUTINTEGER(CONOUT, MIXEDTYPE.INTNUMBER);

      PUTSTRING (CONOUT, #'%REND EXECUTION.%R/');
   END MAIN;
   END DCTBENCHMARK;   ! END MODULE !
```

APPENDIX C
DCT/HOL STUDY BENCHMARK
PROGRAM RESULTS

This appendix contains information generated by running benchmark programs using the candidate languages.

Table C-1. DCT HOL Summary Benchmark Results

Language	Executive Time (Min: Sec)	# Bytes Absolute Object Code(No I/O)	Program Support Environment	Compile Time
Interactive C	:58.5	1286	PDP-11/70 UNIX	:45
Whitesmith C	1:00	2538	CP/M	5:17
FORTRAN-80	4.03	3570	CP/M	1:47
RATFOR	3:43	3925	CP/M	3:01
Pascal/MT	1:36	3298	CP/M	:52
Pascal/Z	2:18	2304	CP/M	3:01
PLI-80	2:30	4514	CP/M	2:17
PLMX	:59	1759	CP/M	7:00
PLZ	2:48 (1)	2165	ZILOG	4:00

(1) Corrected for 2.5 MHZ Z80

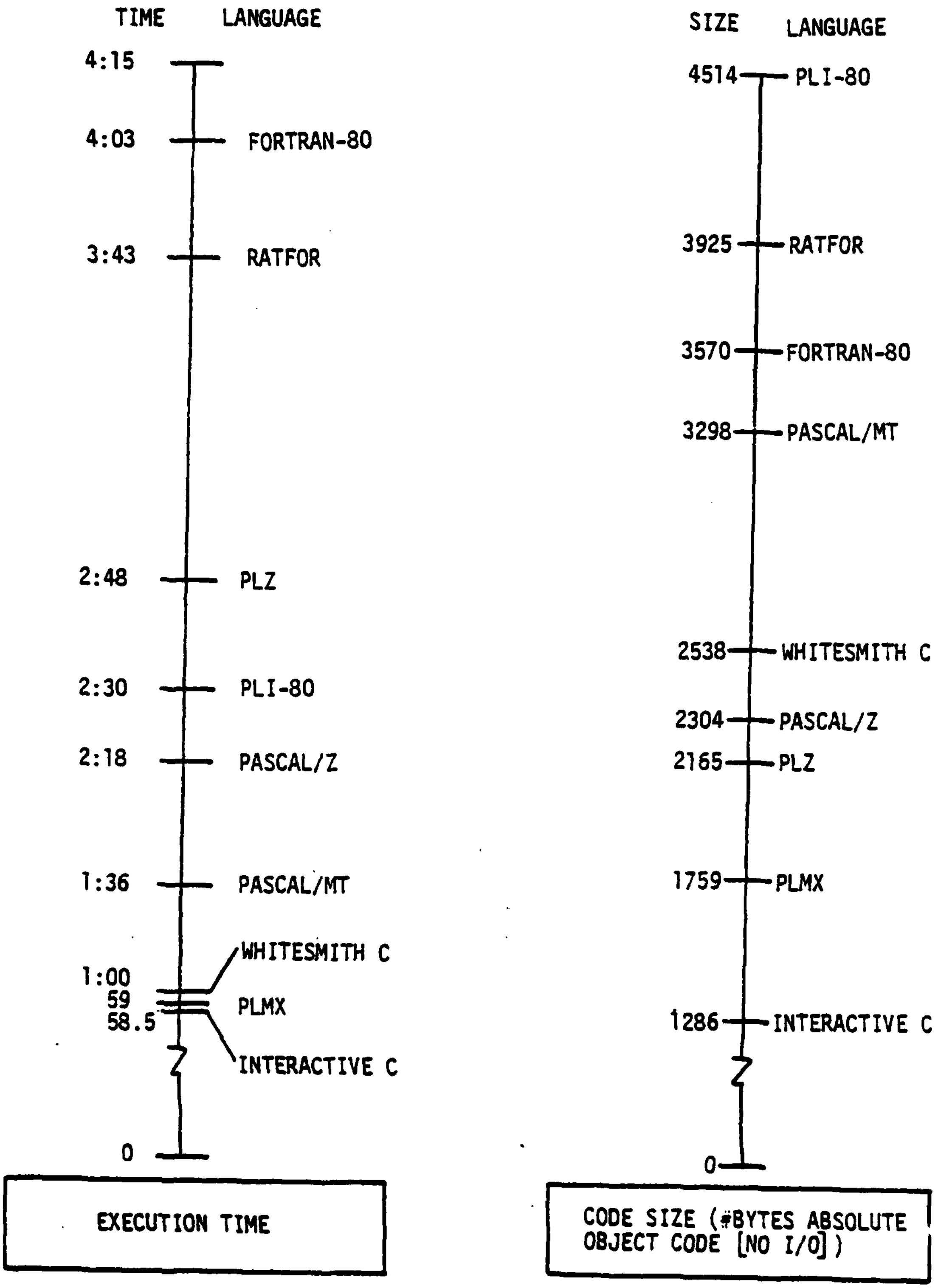

Figure C-1. Benchmark Results

APPENDIX D

DELPHI PHASE 1: DERIVATION OF WEIGHTS

This appendix presents the statistical results of Phase 1 of the Delphi study. The object of Phase 1 was to assign weights to the various language features.

Table D-1. Phase 1 Delphi Study Statistical Summary

	1st Iteration			2nd Iteration			3RD Iteration		
LANGUAGE FEATURE	Average	Std.Dev.	Std.Dev./ Average	Average	Std.Dev.	Std.Dev./ Average	Average	Std.Dev.	Std.Dev./ Average
DATA REPRESENTATION	121.3333	35.1034	.2893	132.5556	31.0447	.2342	137.5	43.6049	.3171
SYSTEMS PROGRAMMING	128.4444	41.6297	.3241	125.2222	30.7481	.2455	124	29.5146	.2380
CONTROL STRUCTURES	115.0000	27.6134	.2401	110.6667	22.4221	.2026	107	21.2393	.1985
PROGRAM SUPP. ENV.	108.6667	37.6132	.3461	101.7778	27.5172	.2704	95	16.4991	.1737
DOCUMENTATION	63.4444	28.2051	.4446	64.7778	22.1460	.3419	74	17.6068	.2379
READABILITY	52.6667	20.8447	.3958	62.3333	20.3531	.3265	67.4	18.7451	.2781
TIME EFFICIENCY	43.4444	19.7934	.4556	50.6667	19.0919	.3768	57.2	17.6874	.3092
SPACE EFFICIENCY	40.2222	16.6717	.4145	51.6667	20.1556	.3901	55.5	16.9066	.3046
EXTENT OF USE	51.0000	29.6564	.5815	47.4444	30.9561	.6525	49	30.0222	.6127
ASSEMBLY LANG. LINKAGE	50.3333	29.8831	.5937	47.7778	22.5875	.4728	45.2	13.2145	.2924
TARGET CPU TRANSPORTABILITY	47.1111	32.6743	.6936	38.2222	20.1791	.5279	31.5	14.3469	.4556
LEARNABILITY	34.7778	18.1437	.5217	35.5556	20.4641	.5755	31.5	13.6646	.4338
NSC 800 (Z80) (INST. SET USE	32.7778	20.4559	.6241	35.3333	20.2176	.5722	31.1	13.4944	.4339
MULTITASKING	24.3333	16.2635	.6684	22.0000	11.7047	.5320	27.5	26.3744	.9591
REENTRANCY & RECURSION	32.0000	12.5897	.3934	27.3333	8.3217	.3045	25	5.2705	.2108
ROMable OBJECT CODE	17.000	5.8737	.3455	15.3333	9.3274	.6083	14.8	8.3106	.5615
COMPILE-TIME EFFICIENCY	20.0000	9.4736	.4737	16.8889	7.2015	.4264	14.5	3.1710	.2187
PSE SOFTWARE TRANSPORTABILITY	17.4444	8.9458	.5128	14.4444	9.7225	.6731	12.3	7.8322	.6368

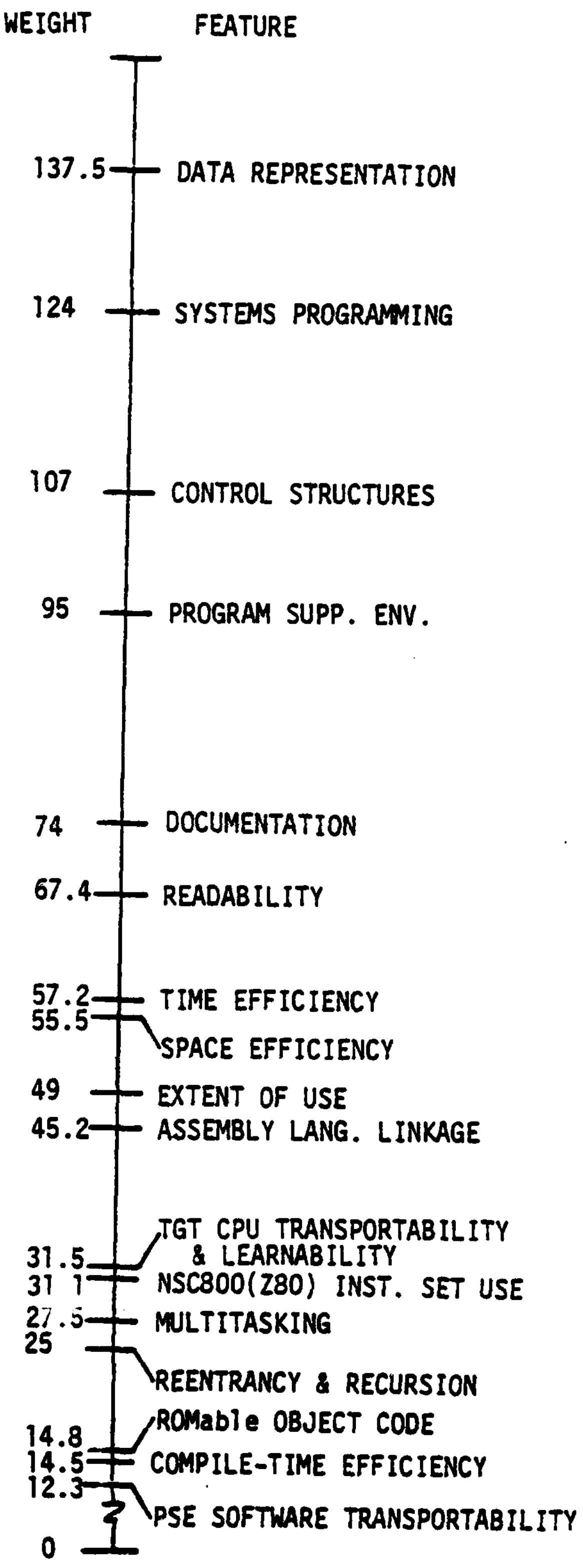

Figure D-1. Ranking of Features

Table D-2. First Iteration DCT HOL Language Feature Response Summary

LANGUAGE FEATURE	RESPONSES - ORDERED LOW TO HIGH									TOTAL	AVERAGE	STD.DEV.	ST DEV./ AVERAGE
SYSTEMS PROGRAMMING	85	90	95	110	120	130	136	190	200	1156	128.4444	41.6297	.3241
DATA REPRESENTATION	75	85	90	100	120	150	150	152	170	1092	121.3333	35.1034	.2893
CONTROL STRUCTURES	80	85	100	100	110	120	130	150	160	1035	115.0000	27.6134	.2401
PROGRAM SUPP. ENV.	70	80	80	88	95	100	130	160	175	978	108.6667	37.6132	.3461
DOCUMENTATION	30	35	46	50	60	65	70	105	110	571	63.4444	28.2051	.4446
READABILITY	15	35	40	50	55	60	60	76	83	474	52.6667	20.8447	.3958
EXTENT OF USE	1	10	35	50	63	70	70	75	85	459	51.0000	29.6564	.5815
ASSEMBLY LANG. LINKAGE	20	21	25	40	45	52	55	90	105	453	50.3333	29.8831	.5937
TARGET CPU TRANSPORTABILITY	5	20	20	29	45	60	65	70	110	424	47.1111	32.6743	.6936
TIME EFFICIENCY	15	20	25	36	55	55	55	60	70	391	43.4444	19.7934	.4556
SPACE EFFICIENCY	15	25	30	30	40	47	50	60	65	362	40.2222	16.6717	.4145
LEARNABILITY	13	15	15	35	35	35	50	50	65	313	34.7778	18.1437	.5217
NSC 800 (Z80) INST. SET USE	6	15	25	30	34	35	35	35	80	295	32.7778	20.4559	.6241
REENTRANCY & RECURSION	15	20	28	30	30	35	35	35	60	288	32.0000	12.5897	.3934
MULTITASKING	10	10	15	15	20	21	25	48	55	219	24.3333	16.2635	.6684
COMPILE-TIME EFFICIENCY	10	12	15	15	18	20	20	30	40	180	20.0000	9.4736	.4737
PSE SOFTWARE TRANSPORTABILITY	2	10	15	15	20	20	20	20	35	157	17.4444	8.9458	.5128
ROMable OBJECT CODE	9	10	14	15	15	20	20	25	25	153	17.0000	5.8737	.3455

Table D-3. Second Iteration DCT HOL Language Feature Response Summary

LANGUAGE FEATURE	RESPONSES - ORDERED LOW TO HIGH									TOTAL	AVERAGE	STD.DEV.	STD.DEV./ AVERAGE
DATA REPRESENTATION.	100	105	110	120	120	120	168	170	180	1193	132.5556	31.0447	.2342
SYSTEMS PROGRAMMING	92	100	100	110	120	130	135	150	190	1127	125.2222	30.7481	.2455
CONTROL STRUCTURES	80	86	100	100	105	120	125	130	150	996	110.6667	22.4221	.2026
PROGRAM SUPP. ENV.	80	80	81	85	90	100	110	130	160	916	101.7778	27.5172	.2704
DOCUMENTATION	40	50	50	55	58	60	70	90	110	583	64.7778	22.1460	.3419
READABILITY	35	38	40	55	70	80	80	80	83	561	62.3333	20.3531	.3265
SPACE EFFICIENCY	30	30	35	40	50	50	75	75	80	465	51.6667	20.1556	.3901
TIME EFFICIENCY	20	35	35	50	52	54	55	75	80	456	50.6667	19.0919	.3768
ASSEMBLY LANG. LINKAGE	21	30	31	40	43	45	60	65	95	430	47.7778	22.5875	.4728
EXTENT OF USE	1	5	30	40	50	63	76	77	85	427	47.4444	30.9561	.6525
TARGET CPU TRANSPORTABILITY	5	15	30	30	40	45	55	59	65	344	38.2222	20.1791	.5279
LEARNABILITY	13	20	20	22	30	35	45	65	70	320	35.5556	20.4641	.5755
NSC 800 (Z80) INST. SET USE	6	25	30	30	30	33	34	50	80	318	35.3333	20.2176	.5722
REENTRANCY & RECURSION	13	15	25	28	30	30	35	35	35	246	27.3333	8.3217	.3045
MULTITASKING	10	12	15	15	20	23	25	30	48	198	22.0000	11.7047	.5320
COMPILE-TIME EFFICIENCY	10	10	10	12	17	18	20	25	30	152	16.8889	7.2015	.4264
ROMable OBJECT CODE	4	5	10	10	14	15	25	25	30	138	15.3333	9.3274	.6083
PSE SOFTWARE TRANSPORTABILITY	2	6	10	10	12	15	20	20	30	130	14.4444	9.7225	.6731

Table D-4. Third Iteration DCT HOL Language Feature Response Summary

LANGUAGE FEATURE	RESPONSES - ORDERED LOW TO HIGH										TOTAL	AVERAGE	STD.DEV.	STD.DEV./ AVERAGE
DATA REPRESENTATION	100	105	110	110	120	120	125	170	180	235	1375	137.5	43.6049	.3171
SYSTEMS PROGRAMMING	85	100	110	110	115	115	130	135	150	190	1240	124	29.5146	.2380
CONTROL STRUCTURES	80	80	90	100	105	110	115	120	120	150	1070	107	21.2393	.1985
PROGRAM SUPP. ENV.	75	80	85	85	90	90	100	100	120	125	950	95	16.4991	.1737
DOCUMENTATION	55	55	60	65	65	75	80	85	90	110	740	74	17.6068	.2379
READABILITY	35	45	50	60	70	80	80	80	86	88	674	67.4	18.7451	.2781
TIME EFFICIENCY	20	45	47	50	55	60	70	75	75	75	572	57.2	17.6874	.3092
SPACE EFFICIENCY	30	35	45	45	50	60	70	70	70	80	555	55.5	16.9066	.3046
EXTENT OF USE	1	5	30	40	50	60	64	70	80	90	490	49	30.0222	.6127
ASSEMBLY LANG. LINKAGE	21	31	40	40	45	45	50	55	60	65	452	45.2	13.2145	.2924
TARGET CPU TRANSPORTABILITY	5	15	25	30	30	35	35	40	45	55	315	31.5	14.3469	.4556
LEARNABILITY	13	20	20	22	30	35	35	40	40	60	315	31.5	13.6646	.4338
NSC 800 (Z80) INST. SET USE	6	25	25	30	30	30	30	30	50	55	311	31.1	13.4944	.4339
MULTITASKING	10	10	12	20	20	23	25	25	30	100	275	27.5	26.3744	.9591
REENTRANCY & RECURSION	15	20	20	25	25	25	30	30	30	30	250	25	5.2705	.2108
ROMable OBJECT CODE	4	5	10	10	14	15	15	20	25	30	148	14.8	8.3106	.5615
COMPILE-TIME EFFICIENCY	10	10	12	15	15	15	15	15	18	20	145	14.5	3.1710	.2187
PSE SOFTWARE TRANSPORTABILITY	2	6	10	10	10	10	10	15	20	30	123	12.3	7.8322	.6368

APPENDIX E
DELPHI PHASE 2: DERIVATIONS OF SCORES AND FIGURES OF MERIT

This appendix presents the statistical results of Phase 2 of the Delphi study. The object of Phase 2 was to assign gross scores and figures of merit to each language based upon the weights of the various language features assigned in Phase 1.

```
AD-A118 811   MARINE CORPS TACTICAL SYSTEMS SUPPORT ACTIVITY  CAMP --ETC  F/G 9/2
              DIGITAL COMMUNICATIONS TERMINAL HIGH ORDER PROGRAMMING LANGUAGE--ETC(U)
              NOV 80  K C SHUMATE, R E SAUER, G E ANDERSON
UNCLASSIFIED  24E005/U-TN-01-VOL-2                                          NL
```

2 of 2

END
DATE
FILMED
9 82
DTIC

Table E-1. Delphi Study--Phase 2 Summary (1st Iteration)

Language Feature	APL	BASIC	C	COBOL	FORT-66	FORT-77	RATFOR	PASCAL	FORTH	PL/I-80	PLMX	PLZ
Data Representation	55	33.6875	89.3750	79.0625	60.885	68.75	82.5	114.4687	59.5787	110	87.0787	110
System Programming	24.8	15.934	110.36	27.28	47.5292	46.5	55.8	80.6000	94.2400	80.6	111.6	80.6
Control Structures	42.8	34.1009	91.8381	53.5	41.2699	72.76	74.9	86.9375	66.3400	89.88	89.1631	85.6
Program Supp. Env.	38	72.0385	76.35	53.2	58.9	57	66.5	74.4135	47.5000	76	82.327	66.5
Documentation	29.6	54.1162	56.1142	61.42	52.6436	60.384	53.65	63.3662	30.2142	57.165	56.7284	62.9
Readability	6.74	37.4878	46.0544	53.92	32.2576	37.744	38.755	59.3929	19.0944	55.268	53.92	57.29
Time Efficiency	11.44	5.434	36.6080	17.16	33.8452	26.455	36.608	35.1322	33.1700	42.9	41.9448	40.04
Space Efficiency	22.2	15.0627	38.85	19.425	36.075	31.9125	40.2375	33.6941	44.8606	38.85	38.85	33.3
Extent of Use	4.9	44.1	28.5817	39.2	37.7986	33.32	30.625	35.5250	9.8000	29.4	37.5634	29.4
Assembly Language Linkage	4.52	10.6536	29.38	17.176	34.352	32.7	32.77	25.6600	31.6400	32.77	9.04	36.16
Target CPU Transportability	12.6	23.94	22.5729	16.5575	24.885	20,475	20.475	15.5263	22.0500	17.325	25.2	16.7989
Learnability	3.15	26.5387	24.6739	21.001	20.0245	19.845	25.2	24.8062	12.6000	21.6562	25.2	26.775
MSC 800 (Z80) Inst. Set Use	12.44	6.9975	14.5112	6.22	15.55	15.55	15.55	16.5856	10.3656	11.6625	21.77	15.55
Multitasking	2.75	13.75	9.6250	5.5	6.1875	8.25	8.25	7.5625	15.5815	9.1657	9.1657	13.75
Reentrancy & Recursion	0	3.3325	21.25	0	2.75	1.6675	1.6675	10.5000	16.2500	16.25	15	13.75
ROMable Object Code	0	6.216	11.1	5.92	10.064	8.88	9.25	10.0640	8.3250	3.33	7.8924	8.88
Compile-time Efficiency	4.35	3.335	11.2375	3.8657	9.4250	8.5187	8.9421	8.9900	9.5700	7.0687	7.25	7.975
PSE Software Transportability	2.46	9.7908	9.84	7.7896	10.4796	8.9175	8.61	10.3320	7.9950	9.225	6.765	3.69
TOTALS	277.7500	416.5157	728.3219	488.1773	534.9217	559.6992	610.2901	663.5567	539.1750	708.5161	726.4585	651.2907

Table E-2. Delphi Study--Phase 2 Summary (2nd Iteration)

Language Feature	Inter-active C	White-smith's C	Microsoft FORT-80	Zilog FORT	UCSD FORT	Cromemco RATFOR	Pascal/MT	Pascal/Z	PL/I-80	PLMX	PLZ
Data Representation	97.2262	92.8125	62.9062	52.25	59.125	68.75	118.8412	104.8437	110.55	69.4375	96.25
System Programming	112.6292	111.6	74.09	71.5	68.2	72.3292	110.5708	79.3600	76.88	110.05	77.5
Control Structures	90.0512	84.7119	53.5	51.36	57.78	75.6597	93.8497	88.0931	75.7881	86.9375	85.6
Program Supp. Env.	71.9625	69.35	68.875	42.75	54.625	69.5115	76	74.1	76.95	68.875	52.25
Documentation	50.32	56.24	54.76	46.25	51.8	51.8	59.2	55.5	53.65	51.8	51.8
Readab.lity	48.1438	47.18	36.2275	39.092	41.788	42.3676	59.2176	57.29	51.6756	49.7075	48.528
Extent of Use	26.46	25.725	36.75	18.0075	18.375	22.05	27.6017	22.05	23.52	40.0183	26.1317
Assembly Language Linkage	42.94	38.42	36.16	21.47	21.47	25.312	35.256	25.312	32.77	43.6903	41.4303
Target CPU Transportability	18.1125	23.625	25.2	22.05	12.6	21.2625	22.68	20.79	24.1511	31.5	25.2
Learnability	24.7496	25.2	24.3338	21.2625	22.68	22.5004	25.6504	24.9386	20.4750	23.625	22.8375
MSC 800 (Z80) Inst.Set Use	26.4350	23.325	27.99	29.545	0	0	26.435	27.99	24.88	13.995	31.1
Multitasking	0	.6875	.9157	.9157	0	0	.6875	.9156	.9157	0	0
Reentrancy & Recursion	21.6675	18.875	.8325	1.25	0	0	20.625	23.3325	18.3325	25	25
ROMable Object Code	0	0	13.32	0	0	0	8.9288	8.9288	7.955	14.8	14.8
PSE Software Transportability	6.9704	9.84	9.84	1.845	5.535	10.1475	9.84	9.4304	9.84	6.9704	1.845
TOTALS*	637.6679	627.5919	525.7007	419.3477	413.978	481.6906	695.3839	552.8749	608.333	636.4065	600.2725

*These totals do not reflect the inclusion of space efficiency, time efficiency, or compile-time efficiency.

Table E-3. Delphi Study--Phase 2 Summary (3rd Iteration--MCTSSA)

Language Feature	Inter-Active C	White-Smith's C	Microsoft FORT-80	Cromemco RATFOR	Pascal/MT	Pascal/Z	PL/I-80	PLMX	PLZ
Data Representation	115.885	110	62.8512	96.25	123.75	129.635	111.9525	79.5437	91.3275
System Programming	109.529	110.5584	51.6584	64.0584	108.6984	103.7384	94.0292	103.3292	98.1708
Control Structures	96.3	96.3	48.9097	81.7694	97.8301	98.1297	87.1194	87.1194	86.3597
Program Support Environment	63.783	78.033	78.033	78.033	78.033	79.952	79.952	78.7075	49.533
Documentation	62.9	54.8784	57.72	59.94	61.42	62.9	54.02	51.8	58.9484
Readability	53.92	54.877	42.3609	49.5861	60.66	60.7543	55.8476	55.4567	56.8114
Time Efficiency	57.2	55.77	13.768	15.0036	34.8576	24.2471	22.308	56.7138	19.917
Space Efficiency	55.5	28.128	19.9911	18.1818	21.6394	30.9745	15.8119	40.576	32.967
Extent of Use	28.5817	33.8884	41.2384	38.3817	33.8884	38.3817	33.4817	26.1317	35.525
Assembly Language Linkage	38.0945	29.7009	35.1882	27.4409	30.3472	37.1273	34.5418	34.8672	36.16
Target CPU Transportability	5.2478	12.6	23.0989	19.9489	21.105	11.8125	18.9	25.7229	6.615
Learnability	24.7495	25.4236	24.3432	24.2991	28.35	28.35	21.5995	24.7495	24.9732
MSC 300 (Z80) Inst. Set Use	26.9512	3.6262	3.6262	0	25.3962	23.0637	3.6262	27.99	27.99
Multitasking	0	0	0	0	0	0	0	0	0
Reentrancy & Recursion	24.5825	24.5825	0	0	22.5	20.415	2.5	0	25
ROMable Object Code	7.8218	14.5524	7.8928	5.92	14.5528	13.0728	7.6456	13.32	14.5528
Compile-time Efficiency	14.5	2.0575	6.0987	3.6047	12.5483	3.6047	4.7632	1.5529	2.7187
PSE Software Transportability	5.3295	9.84	11.172	11.172	11.2741	11.2742	11.2741	10.455	5.3295
TOTALS*	663.6755	658.3608	488.0929	556.7995	717.8052	718.6067	616.4896	619.1928	617.2963
FOM	790.8755	744.8101	527.9595	593.5896	786.8505	777.433	659.3727	718.0355	672.899

*Total without space, time, compile-time efficiency.

Table E-4. Delphi Study--Phase 2 Summary (3rd Iteration--NOSC Included)

Language Feature	Inter-active C	White-smith's C	Microsoft FORT-80	Cromemco RATFOR	Pascal/MT	Pascal/Z	PLI-80	PLMX	PLZ
Data Representation	115.0875	111.82	65.6975	95.3837	123.75	127.5725	110.8663	79.5437	91.3275
System Programming	110.9056	112.282	55.0188	68.2	103.9988	94.86	92.1196	103.3292	98.1708
Control Structures	95.979	93.9246	49.9369	82.925	95.6259	94.7592	86.9375	87.1194	86.3597
Program Support Environment	67.7635	75.3635	75.468	75.4015	75.468	75.05	76.6745	78.7075	49.533
Documentation	60.8428	55.0338	56.5582	58.5858	57.6164	56.8912	56.1142	51.8	58.9484
Readability	52.572	53.1719	38.3438	49.2829	59.5344	59.6086	53.92	55.4567	56.8114
Time Efficiency	57.2	55.77	13.768	15.0035	34.8577	24.2471	19.11	56.7152	19.917
Space Efficiency	55.5	28.1218	19.99	18.1818	21.4174	30.9745	15.8119	40.576	32.967
Extent of Use	28.3122	34.6038	39.2588	35.7014	32.34	33.1975	29.4	26.1317	35.525
Assembly Language Linkage	38.8268	30.5868	35.9069	29.0952	30.6366	34.9034	35.3148	34.8672	36.16
Target CPU Transporta-bility	8.6625	12.7953	22.05	19.3504	21.1491	13.586	18.4496	25.7229	6.615
Learnability	23.94	24.6395	23.1336	24.214	27.1246	27.1247	21.2625	24.7495	24.9732
NSC 800 (Z80) Inst. Set Use	27.2654	4.0803	4.0803	0	25.7137	19.2447	3.110	27.99	27.99
Multitasking	1.375	1.375	0	0	0	0	0	0	0
Reentrancy & Recursion	24.1675	24.0625	.8925	0	20.535	19.22	5.000	0	25
ROMable - Object Code	9.62	14.1651	6.7888	5.0734	13.505	12.6851	7.6472	13.32	14.5528
Compile-time Efficiency	14.5	2.0575	6.0987	3.6047	12.5483	3.6047	4.6732	1.5529	2.7137
PSE Software Transportability	4.578	9.3013	9.455	11.1573	9.6555	9.61	9.6641	10.455	5.3295
TOTAL*	669.8978	657.2052	482.5891	554.3706	694.653	678.3129	607.4803	619.1928	617.2963
FOM	797.0978	743.1545	522.4458	591.1606	763.4764	737.1392	647.0724	718.0369	672.899

*Total without space, time, compile-time efficiency.

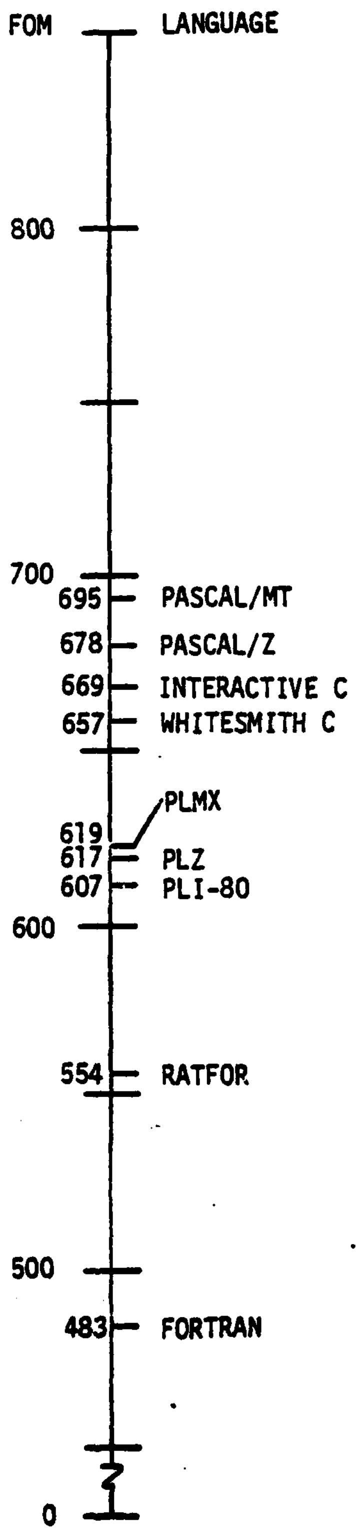

Figure E-1. Figures of Merit without Benchmark Values

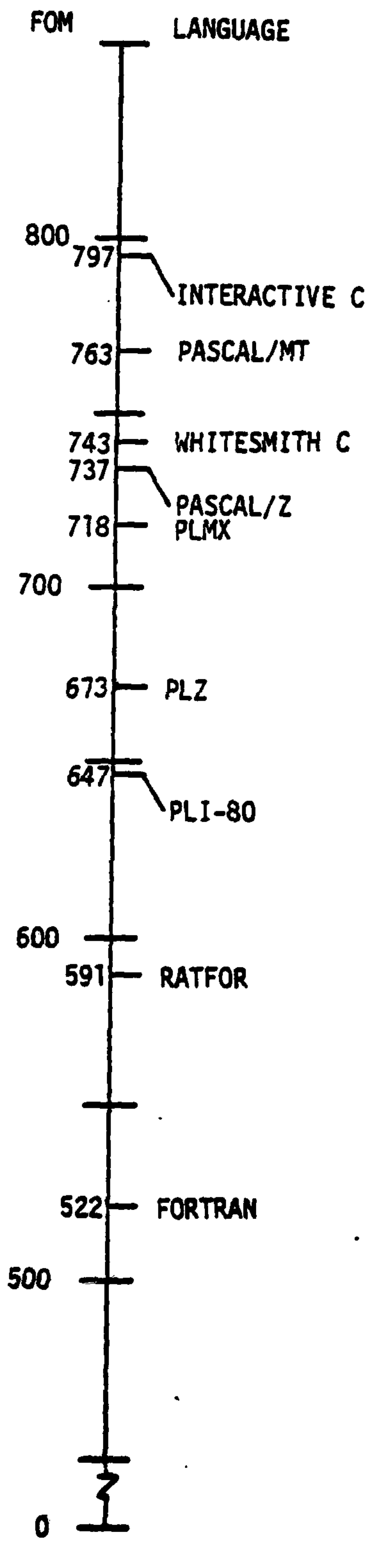

Figure E-2. Final Figures of Merit

ATE

LMED

8